DOCUMENT RESUME

ED 211 062 IR 009 880

AUTHOR Crawford, Stuart
TITLE A Standard's Guide for the Authoring of Instructional Software. Reference Manual Volume III.
INSTITUTION JEM Research, Victoria (British Columbia).; Victoria Univ. (British Columbia).
PUB DATE 81
NOTE 145p.; For related document, see IR 009 758.
AVAILABLE FROM JEM Research, Discovery Park, PO Box 1700, Victoria, B.C. V8W 2Y2 Canada ($75.00).

EDRS PRICE MF01/PC06 plus Postage.
DESCRIPTORS *Computer Assisted Instruction; Computer Graphics; *Computer Programs; Content Analysis; Evaluation Criteria; Flow Charts; Guidelines; Instructional Design; Learning Theories; *Material Development; *Microcomputers; *Programing; Programing Languages
IDENTIFIERS *Apple II; *Authoring Aids

ABSTRACT

This guide gives step-by-step directions for authoring computer-assisted instruction materials as well as advice on planning, designing, coding, documenting, and evaluating computer assisted courseware. The importance of prospective courseware designers having a sound background in both educational theory and practice is also discussed, and various learning theories and their applications for computer courseware design are examined. Appendices provide information on choosing a computer language and on using graphics on the Apple II microcomputer. Courseware evaluation instruments are also included, and 65 references are listed. (LLS)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

jem research
Joint Educational Management Research

ED211062

# A STANDARD'S GUIDE FOR THE AUTHORING OF INSTRUCTIONAL SOFTWARE

## REFERENCE MANUAL VOLUME III

by: Stuart Crawford

IR009880

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
W. Tannent
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

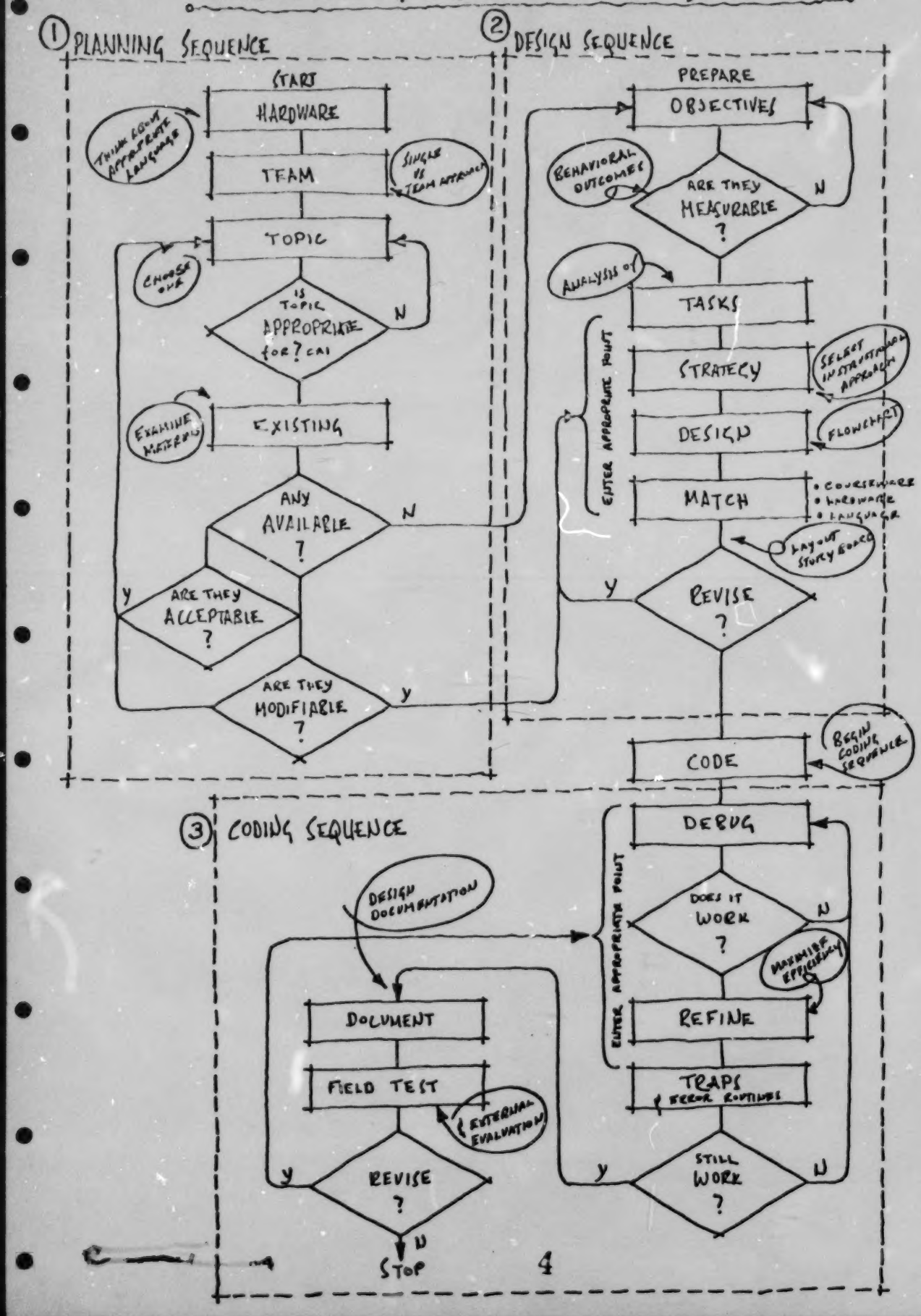


Discovery Park
University of Victoria
Victoria, B.C. V8W 2Y2
Tel. (604) 477-7246

TABLE OF CONTENTS

| | PAGE |
|---|---|
| INTRODUCTION | 1 |
| LEARNING THEORIES AND COURSEWARE DESIGN | 4 |
| PLANNING SEQUENCE | 15 |
| DESIGN SEQUENCE | 19 |
| CODING SEQUENCE | 48 |
| DOCUMENTATION AND EVALUATION SEQUENCE | 52 |
| APPENDIX 1: | |
| CHOOSING A LANGUAGE | 60 |
| APPENDIX II: | |
| GRAPHICS ON THE APPLE II | 83 |
| APPENDIX III: | |
| COURSEWARE EVALUATION INSTRUMENTS | 97 |
| REFERENCES | 126 |

# AUTHORING INSTRUCTIONAL SOFTWARE



# INTRODUCTION

## INTRODUCTION

The role of the computer in education is a topic that is frequently tackled in the educational literature. Computer assisted instruction (CAI) is generally believed to have five major advantages:

1. As opposed tc traditional modes of instruction, the learner is rarely able to be a totally passive receiver of information. When using a CAI program, the student is generally an active participant in the learning process (Chambers and Sprecher, 1980).

2. CAI has the potential to be a highly individualized method of instruction. Students can proceed at their own pace and can respond to questions without the fear of embarassing failure. In addition, students can receive important individualized feedback regarding progress being made (Dence, 1980).

3. Students can simulate experiments or explore situations which would be too dangerous, expensive, difficult or time consuming to undertake in the real world (Gaede and Singletary, 1979).

4. When computers are used as an aid to instruction, classroom teachers can become "facilitators" of instruction rather than lecturers. In addition, teachers generally have more free time to devote to dealing with students in a more personal and meaningful manner (Chambers and Sprecher, 1980).

5. The computer evaluates a student's performance in a completely objective manner, and the possibility of unconscious teacher bias is eliminated (Clement,1981).

In spite of the potential advantages listed above, many educators are still hesitant to use the computer in the classroom. This resistance is due, at least partially, to CAI's inauspicious beginnings.

> Computer-based methods developed during the 1960s and 1970s left an image in the minds of many educators that was less than favourable. The powerful resource was often used as an expensive electronic page-turner. Instructional materials were often developed by technicians with little expertise in the principles of instructional design (Clement, p.31).

In addition, when the Educational Testing Service evaluated the PLATO and TICCIT projects (the combined cost of the projects was approximately $60,000,000), "both were tacitly evaluated as failures." (Hirschbuhl, 1980).

The problem is by no means rectified today. In fact, "much of what passes for courseware today neither helps to develop higher cognitive skills nor challenges learners to use higher order learning strategies" (Caldwell,p.7). Some authors are somewhat less diplomatic, and simply state that many of the available educational programs are "junk" and are "virtually devoid of any instructional value and in many cases are acting as deterrents to the widespread acceptance of CAI" (Gleason,pp.11,12).

Unlike the suggestions made by some manufacturers and authors, the development of courseware is not "...as easy as 1 2 3" (Garraway,p.20). The development process is, in fact, costly, difficult and time consuming. It is often stated that two hundred hours of development time are required for the production of one hour of instructional time, and some authors suggest that a cost estimate of $10,000 per instructional hour is not excessive (Gleason, 1981). Furthermore, the development of courseware is a financially risky proposition due to the current difficulties with copyright laws and software piracy (Becker, 1981).

Clearly, much work has to be done in order to improve the overall quality of available courseware. Some authors take a somewhat pessimistic view, suggesting that "unlike hardware, courseware is not likely to improve in the immediate future" (Sugarman,p.82). However, regardless of the view taken, it seems certain that before courseware can

improve, "instructional design expertise must be wedded to computer expertise" (Gleason,p.12).

The problem of improving the quality of courseware is of extreme importance to the field of computer education. In fact, "there is agreement among instructional computing experts that the future success or failure of microcomputers as a medium of instruction...hinges largely on the availability of quality courseware" (Robleyer,p.243).

# LEARNING THEORIES AND COURSEWARE DESIGN

## LEARNING THEORIES AND COURSEWARE DESIGN

As suggested in the introduction, it is essential that prospective courseware designers have a sound background in both educational theory and practice. In particular, a knowledge of theories regarding the process of learning is important. In fact, as Gagne and Briggs (1974) point out, "designing instruction must be based upon knowledge of how human beings learn." (p.19) A designer must know how learning takes place if the courseware is to be effective. The purpose of this chapter is to introduce the reader to some theories of learning and to relate these theories to the practice of courseware design.

The psychology of learning is a relatively new science, having only been studied for about a century. As such, the process of learning is not fully understood. It is perhaps useful to begin this discussion with a working definition of the learning process. One definition (Silverman, p.198) describes learning as, "a process in which past experience or practice result in relatively permanent changes in an individual's repertory of responses". It is clear that learning involves changes in behaviour, but it is unclear whether this behavioural change is external and easily observable, or whether the behavioural change is internalized (thinking, considering) and not so easily observable. Learning theories fall into two main classes which take into account the problem above. The stimulus-response theorists emphasize the notion of learning as observable and somewhat predictable. The cognitive theorists, on the other hand, feel this approach is too mechanistic and place emphasis on the role of perception in learning. We will first examine the stimulus-response approach, and then explore the cognitive theories.

### STIMULUS-RESPONSE ASSOCIATION

Association was one of the earliest processes of learning to be formally identified. It is by the association of stimuli with appropriate responses that young children learn to attach names to objects in their

environment. One of the earliest such associations occurs when infants learn to attach the word "mother" to the image of their mother. A drill and practice CAI program that presented the user with various geometrical shapes could be considered an example of S-R Association. In this type of program the user could, for example, learn to associate the word "circle" with the appropriate set of images, or associate the phoneme "ay" with the letter "a".

The two major classes of S-R Association described below are often used as models for studying the process of learning.

Classical Conditioning:

This type of conditioning (also known as respondent or Pavlovian conditioning) is characterised by learner responses which are reflexive in nature (e.g. not under the learner's voluntary control). This type of conditioning is perhaps best illustrated by the, now classic, Pavlovian experiments with dogs. The dogs were initially presented with the sound of a struck tuning fork (the conditioned stimulus), but they quickly became accustomed to this neutral stimulus and began to ignore it. The subjects were then presented with food powder (the unconditioned stimulus), and the food powder elicited the reflexive response of salivation (the unconditioned response). After many trials in which the tuning fork is presented in association with the food powder, it was found that the tuning fork alone was capable of eliciting the salivation response. Since the response of salivation has become learned, it is now called the conditioned response.

Since most existing courseware is designed to encourage learning above the reflexive level, classical conditioning appears to have little to do with CAI. However, some of the popular computer games appear to contain some elements of classical conditioning. The Space Invaders and Asteroids games are two possible examples of this, as described overleaf.

Conditioned Stimulus:

Throughout the duration of the games, a repetitive buzzing or clicking sound occurs. This sound increases in rapidity as the game progresses from easy to difficult situations. This type of sound is one which, if initially presented without any association to a game, could be considered a neutral stimulus.

Unconditioned Stimulus:

Many small, fast-moving asteroids present a strong stimulus to the player.

Unconditioned Response:

Increased stress, attentiveness and (probably) more rapid heartbeat are the traditional reflexive responses.

Conditioned Response:

It appears reasonable to suggest that after playing the game several times, the player soon learns to associate the more rapid occurrence of the sound with the increased difficulty level of the game. In this way, the changes in rapidity of sound occurrence alone would be sufficient to elicit the desired response.

If the above premise is indeed true, it may well be that employment of classical conditioning techniques in courseware is one method of controlling user attentiveness and interest. At any rate, the stimulating effect of computer games should be carefully studied. One author (Clement, p.29) notes that,

> We need to understand what it is that will keep people entranced by computerized games for hours at a time. Very often they are developing complex skills of planning and strategy that instructional designers would find hard to emulate by traditional methods.

It is interesting to note that some researchers (Staats and Staats, 1958) have found that attitudes can be learned as a result of classical conditioning techniques. These researchers demonstrated that attitudes towards nationalities could be successfully conditioned. It is important to note, however, that it was only possible to alter instant, non-thinking, reflexive attitudes. Classical conditioning was not able to alter well-formed and deeply ingrained attitudes. The relevance of this research to CAI development should not be ignored, however, since it has been demonstrated that "student attitudes toward, and familiarity with, CAI have an effect on CAI's usefulness" (Dence, p.51) and that "poor attitudes on the part of instructors have actually resulted in covert and in some cases overt sabotage to the computer-aided learning process" (Clement, p.28).

Instrumental Conditioning:

This type of conditioning (also known as operant conditioning) differs from classic conditioning in that it involves responses that can be controlled by the individual, as opposed to purely reflexive responses. Since the response in question more closely approaches a cognitive level, this type of conditioning holds more promise for application towards courseware design.

An example of operant conditioning would be a situation in which a student solves a problem correctly (produces the desired response) and this response is instrumental in eliciting praise (reinforcement) from the teacher. Since this reinforcement increases the tendency of the student to make the correct response (answer the question correctly) in subsequent trials, the reinforcement is positive (it is a reward). Reinforcement plays a major role in this type of conditioning. In CAI development it is therefore crucial that incorrect responses are not positively reinforced. Situations in which incorrect answers yield flashing lights, sinking ships, explosions or other such interesting events should therefore be avoided.

It is important for the courseware designer to distinguish between primary and secondary reinforcement.

Primary reinforcers are effective purely on their own merit. Pleasing colour patterns or graphics sequences are examples of primary reinforcers. Secondary reinforcers, on the other hand, are only capable of generating a reinforcing effect if the student has previously associated them with other, primary reinforcers. For example, spoken praise will only act as reinforcement if the student has formed an association between it and a physical demonstration of praise (such as a hug). In a similar fashion, good grades will be effective reinforcers only if the student has formed an association with previous occurrences of spoken praise (which, in itself, is previously linked to physical praise).

Reinforcement in CAI need not follow every correct response, (continuous reinforcement) but can, instead, be applied discontinuosly by means of various reinforcement schedules. In fact, research has demonstrated that while learning takes longer to occur when discontinuous reinforcement is applied, it is less easily extinguished than if the learning had occurred as a result of continuous reinforcement. Some different reinforcement schedules are described below.

Fixed Ratio:

Reinforcement occurs after a fixed number of correct responses. For example a graphics display could occur after every ten correct responses, in which case the reinforcement schedule would be described as 10:1. It appears that high ratios (such as 10:1) encourage the learner to make many responses.

Fixed Interval:

Reinforcement occurs after a fixed time interval. For example, reinforcement could occur following the first correct response to be generated after the specified time interval.

Variable Ratio:

This type of schedule maintains an average reinforcement ratio for the duration of the learning session. However, the immediate ratio varies from question to question. This type of schedule tends to encourage a high level of attention for a long period of time, since the learner never knows just when the reward will arrive. It is best to vary the ratio randomly, or at least to use a series complicated enough to prevent learner prediction of reinforcement occurrences.

Variable Interval:

This type of schedule maintains an average time interval for reinforcement throughout the learning session. However, during the session, reinforcement occurs at varying time intervals. This schedule tends to encourage student work at a high and constant rate.

It should be noted that fixed schedules characteristically show pauses or declines in learning immediately following the occurrence of reinforcement. This phenomenon does not occur when variable schedules are implemented. According to Silverman

> the most effective teaching application of this principle probably happens when the learner receives continuous reinforcement at first, then is gradually shifted to an intermittent reinforcement schedule. (p.734)

Although classical conditioning (associated with I. Pavlov) and instrumental conditioning (associated with B.F. Skinner) are the most often cited association theories, other such theories do have relevence to courseware design. For example, contiguity theory (associated with E.R.Guthrie) tends to diminish the importance of reinforcement in learning. This theory suggests that the learner will simply associate a stimulus with a response and that once the association has been made, the stimulus will then naturally elicit the response. Of importance to

CAI developers is the concept of recency. This concept suggests that the strength of an association is strongest at the first coupling of stimulus and response, and that forgetting comes about as a result of the presence of competing and interfering stimuli.

Behaviour theory (associated with C.L. Hull) is built on the concepts of Thorndike (trial and error learning) and Pavlov. Hull suggests that learning occurs only when reinforcement is able to reduce bodily needs. Drive reduction is therefore the basis of reinforcement, and reinforcements which do not reduce drive are consequently useless. In this fashion, a CAI program which reinforced every correct response would, according to behaviour theory, be ineffective. Since the learner would gradually become accustomed to the reinforcement, it would eventually no longer serve as an aid to drive reduction (a means to satisfy the student's need for praise) and would become useless.

## COGNITIVE THEORIES OF LEARNING

These theories of learning are based on the concepts of Gestalt psychology, and do not accept the S-R theorists view of the learner as a mechanistic and predictable organism. Cognitive theorists tend to be interested in the different ways in which learners perceive various stimuli to be different from each other.

Purposive Behaviourism (based on the works of E.C. Tolman) suggests that the directive stimulus to learn comes from within the learner. Learning is viewed as a process of understanding in which the learner's activities are always directed towards a goal. As stated by Huntingdon

> behaviour is goal directed and the goals of the instructional system must be compatible with the goals of the learner in order for learning to take place at a reasonable rate. Lesson writers may lure learners into learning by manipulating events that alter learner expectancies. (p.120)

Gestalt Theory suggests that learners solve problems by means of insight, and that problem solution tends to reduce learner tension and anxiety by allowing the learner to more effectively understand sensory input. Gestalt theorists view learning as a change in perception and therefore, "the construction of images in verbal or pictorial form must take student reactions into account in order to be successfully used" (Huntingdon, p.120).

Although it appears that major differences exist in learning theory (S-R theorists as opposed to cognitive theorists), these differences can be viewed more as a continuum of different views than as a true dichotomy. Much sharing of information and views occurs and, hopefully, "the principles on which theorists agree provide useful guidelines for both student and teacher" (Silverman, p.230).

## FACTORS AFFECTING LEARNING EFFICIENCY AND RETENTION

Now that we have explored some of the major theories of learning, we can examine some research findings that indicate how the efficiency of the learning process may be increased.

### Feedback:

It is important to distinguish between the feedback of reinforcement (already discussed) and informational feedback. Informational feedback simply provides the learner with information about performance and progress results. A pleasant graphics display following a correct response could be described as positive reinforcement, whereas telling the learner that "You have scored 97% on this test, with an average response time of 12 seconds. All of your errors involved incorrect calculation of square roots." is considered informational feedback. Of course, if the learner is suitably motivated, informational feedback can also serve as an important source of positive reinforcement.

Studies indicate that students learn more from courseware when they are provided with feedback than they do when no feedback is given. (Tait, Hartley, and Anderson, 1974)

Informational feedback may be provided immediately following a response, or the feedback may be delayed. Students, in traditional classroom testing, are generally provided with delayed feedback, with the length of the delay usually being dependent on how long the teacher needed to mark the tests. It has been demonstrated, however, that the efficiency of learning is often increased when students are provided with test scores immediately following completion of the test. While this immediacy is often difficult to accomplish with traditional testing procedures, computer assisted testing can easily provide students with "instant" feedback. In fact, it has been stated that immediate feedback is, "one of the most powerful tools in the arsenal of instructional design" (Kulhavey, p.526).

There do, however, seem to be some situations in which learning efficiency is unaffected or even increased by delaying feedback. Sturges (1969) noted that if retention was tested immediately following the presentation of feedback, then delaying the feedback did not affect learning. Other researchers (Rankin and Trepper, 1978) observed that delaying feedback in a CAI program increased learner retention.

In addition to the timing of feedback, it is also important to take note of the type of feedback to be used. For example, one researcher (Sturges, 1969) examined the effects of two different types of informational feedback in a multiple choice exam. One type simply provided students with the correct answer to each question that was answered incorrectly. The second type showed the student all the possible answers, with the correct response underlined. When a retention test was given seven days following the exam, it was found that those students that had been provided with only the correct answer retained significantly less than the students who had been provided with the more detailed feedback.

Duration:

Various studies indicate that the efficiency of learning is affected by both the duration of the learning session and by the distribution and length of rest breaks.

Learning seems to be facilitated when brief rest periods are distributed throughout the learning session. However, unless the rest breaks meet the needs of the learner, they may prove to be ineffective, or even to interfere with the learning process. Some learners may benefit from a brief rest, whereas others may resent the interruption. It is important not to introduce rest periods too frequently, as the breaks may cause the instructional material to appear artificially disjointed. Furthermore, the instructional periods should be long enough for the learner to absorb the necessary information.

Retention:

Retention can be increased if material is learned beyond the point of mastery. One study (Postman, 1962) examined a group of students who learned material to the point of mastery, and then overlearned by 100%. The results indicated that overlearning not only increased retention, but reduced the amount of time required to relearn the material at a later date. Another researcher (Krueger, 1929) concluded that

> A certain degree of overlearning, at least 50%, is highly economical from the standpoint of retention for intervals of 2 to 28 days, and the larger the interval the greater is the economy. Further increases of overlearning, however, proved to be uneconomical for most intervals. (p.78)

Motivation:

Although the topic of motivation has been discussed earlier in this chapter, it is perhaps useful to examine some aspects of motivation more closely. Motivation begins when a learner experiences a need (perhaps a need for

praise or encouragement). This need then activates a drive (sometimes referred to as a motive) and the learner then exhibits some form of behaviour. The behaviour may then elicit reinforcement, often in the form of need-reduction. Since it is a common observation that high levels of motivation increase the efficiency of the learning process, instructional designers should bear in mind that, "motivation is more than non-directed affective arousal" (Spitzer, p.67). The mere presentation of information on a video screen, no matter how interesting or attractive is somewhat limited in effectiveness in terms of motivational design. "Simulations and gaming contests, in contrast, allow the instructional designer great latitude and opportunity for incorporating motivational principles in order to maximize student motivation" (Spitzer, p.67).

In summary, it should be clear that although the process of learning is complex and not fully understood, it is critically important that prospective courseware designers be aware of the theoretical aspects of the learning process.

PLANNING SEQUENCE

## PLANNING SEQUENCE

As was mentioned in the introductory chapter, many of the CAI programs available on the market today are of poor quality, and are ineffective as aids to instruction. One probable explanation for this is that these programs are not the result of a carefully planned and systematic approach to courseware design. The importance of careful planning is emphasized by Schuyler, who states that "the planning stages of courseware creation are, of course, the most critical" (p.29). An organized and well thought out approach to courseware design will not only result in a superior end product, but the overall development time involved will probably be minimized. There are many steps to consider in the planning stage of courseware development, and each will be considered here under a separate heading.

### THE DESIGN TEAM

As already suggested, the development of educationally sound courseware is an extremely difficult, time-consuming and expensive process. Most authors suggest that the design process is "well beyond the capabilities and resources of individuals and even small groups of teachers" (Gleason, p.12) and that "the notion that computer-based materials can be produced by anybody, completely by themselves, is an archaic concept" (Bork, p.20). It appears that courseware ought to be designed, not by individuals, but by a team of specialists, in which different tasks are assigned to different team members. For example, "building objectives is a task that should be completed by experienced educators using an appropriate procedure" (Huntingdon, p.141), whereas coding tasks should be tackled by experienced programmers. Bork points out that teachers ought not to be expected to carry out the programming chores, in the same way that authors of books are not expected to run a printing press. In addition to insuring that the necessary expertise is available for the design process, the existence of a team also insures that "the biases and errors of one individual are not reflected in the final product" (Roblyer, p.245). A design team

should consist of an instructional designer, a programmer, at least one subject area specialist, and a team leader. Team leaders should familiarize themselves with the rudiments of instructional design and computer programming, so that they "can communicate effectively with design specialists and programmers" (Peters and Johnson, p.10).

It should be mentioned that not all researchers agree that courseware, in order to be effective, must be produced by design teams. At least one author (Gerhold, 1980) takes a very different view.

> Many would argue that teams of professional teachers, curriculum designers, educational psychologists, graphic designers and computer programmers produce higher quality CAI courses than the teacher-programmer could produce. Even if this were true, the point would be moot if the produced materials were unused because of teacher resistance. Regardless, the lack of proven algorithms for 'good' teaching and the lack of devices for measuring the quality of teaching materials...precludes resolution of that argument. We find that teachers know many clever ways to teach. If teacher-programmers are given a suitable CAI language, and if they are exposed to sample materials which illustrate the true capabilities of the computer, then the teacher-programmers produce very useful and appealing CAI courses. Our opinion is that some of these materials are actually superior to the commercially produced materials we have seen. The central point is that they are good enough to be effective, and they will be used (p.20).

It is certainly true that many teacher produced materials are superior in quality to commercially produced packages. However, it is not unreasonable to suggest that the poor commercially produced materials were not the products of a design team, and that teachers with only a little programming experience can produce better courseware than programmers with only a little instructional design experience. At any rate, the same author later concedes that, "the materials produced by teacher-programmers are noticeably inferior to professionally produced materials in

one respect, graphic design." (p.21)

One very effective design team would be one in which classroom teachers joined forces with both computer professionals and the instructional design experts of the educational publishing world.

## TOPIC CHOICE

Once the design team has been selected and assembled, a courseware topic should be carefully selected. It is extremely important to note that not all topics are appropriate for CAI development. Research indicates that topics in the sciences, mathematics and foreign languages are well suited to the development of effective CAI materials. (Koch, 1973) Music also appears to be a subject area in which CAI programs can be very effective, with some studies stating that, "students (were) completing music theory courses in less time and at higher mastery levels" (Hirschbuhl, p.46). In a general sense, computerized games and simulations, regardless of subject area, appear to have great potential for use in educational settings (Dennis, Muiznieks and Stewart, 1979). Of course these are just general guidelines, since topics within many other subject areas can be effectively taught using CAI techniques. Prospective courseware designers should note, however, that some topics may be taught more effectively by traditional methods of instruction and therefore, "it is important that we don't attempt to 'computerize' all experiences" (Kehrberg, p.105).

Once a topic has been carefully selected , it is important to spend some time carefully examining existing courseware which covers related topics. This step is important for two reasons. Firstly, it will enable the instructional designers to familiarize themselves with the capabilities of the computer. In this way, they can later insure that the courseware is designed to capitalize upon those capabilities. Secondly, it may be that effective CAI materials already exist for the chosen topic. There is little point in "reinventing the wheel", and the expense and energy required for courseware development could then be directed towards other topics which had not yet been explored. However, one author (Gerhold, 1983) suggests

that

> Reinvention of wheels is inherent in any learning process. Unless we expect teachers to be willing to use computers as mysterious black boxes, reinvention and duplication are unavoidable. Furthermore, why is it more rational or virtuous to create four alternative courses for a particular subject than four hundred or four thousand? We chose to view CAI materials as dynamic textbooks, and all good teachers always want to modify textbooks for their own purposes. (p.20)

If it turns out that some of the existing courseware is useful, but not quite what the design team desires, then modification of the existing courseware may be possible. If this option is chosen, however, courseware developers must exercise caution, and always be fully cognizant of possible copyright violations.

DESIGN SEQUENCE

## DESIGN SEQUENCE

The initial planning stage of courseware development may be considered to be complete when a topic and target learner group have been finally selected. At this point, the courseware development process enters the design sequence.

### SPECIFICATION OF GOALS AND OBJECTIVES

Once an appropriate topic has been chosen, carefully composed goals and objectives must be prepared. It is perhaps useful at this time to distinguish between these terms.

Instructional goals are typically stated in broad, general terms which indicate the desired outcomes which would result from learners using the instructional materials. These goals do not provide detailed information regarding specific competencies learners are expected to acquire, they simply provide the instructional designers with a general direction in which to proceed. A statement indicating that students will "learn how to draw maps" is sufficiently broad to be considered an example of a goal statement.

In comparison, instructional objectives are much more narrow in scope, and should be stated in behavioural terms. A good instructional objective should clearly indicate the specific observable behaviour which is the desired outcome of the learning process. Mager (1962) states that instructional objectives should state what the learners will be able to do, under what conditions the learner will achieve the objective, and what the criteria for evaluation will be. Because of the emphasis on observable behaviour, instructional objectives are often called behavioural objectives.

Goals and objectives may fall into the cognitive, affective or psychomotor learner domains. However, most educational computer facilities are not able to deal with

the psychomotor domain, and it is often difficult to accurately observe and evaluate behavioural changes in the affective domain. As such, behavioural objectives for CAI are predominantly written with the cognitive domain in mind. Some authors express concern about this emphasis on the cognitive domain, stating that "computer-based instructional materials...lack values clarification, humanistic touches and affective thema", and that "computer based instruction has got to be more" (Donoian, p.85).

Behavioural objectives in the cognitive domain can, according to Bloom's taxonomy, be arranged in six levels of hierarchy:

I. Knowledge

II. Comprehension

III. Application

IV. Analysis

V. Synthesis

VI. Evaluation

It is often useful, from a design point of view, to prepare behavioural objectives with this hierarchy in mind; designing the instructional process so that it smoothly flows from one cognitive level to another. More information about this taxonomy may be found in the, now classic, 1956 paper by Bloom et al.

Unless one wishes to explore the realms of artificial intelligence, it is quite difficult to design CAI programs capable of evaluating behavioural changes in the higher cognitive levels. Because of this problem, most commercially available courseware deals only with the knowledge level. Huntingdon (1979) suggests that "the success of CAI may depend on how quickly and thoroughly we learn to write lessons requiring these higher levels of cognitive functioning" (p.130).

Some instructional designers deal with behavioural objectives on two different planes. The first plane deals

with the objectives that,when met, indicate that the learner has achieved at least the minimum desirable level of competence. These objectives are called core or minimal level objectives, and students meeting these objectives are considered to have a reasonably good chance at successfully completing the next level in the instructional sequence. The second plane involves those objectives which, when met, indicate that the learner has achieved a level of competence which insures a much higher probability of success during the next instructional sequence. These objectives are known as depth or desirable level objectives.

Stating behavioural objectives aids both the instructional designer and the learner. The designer is provided with a means of directing the design process towards an explicitly stated end result, and the learners understand what is expected of them, and can adjust their study efforts accordingly. The importance of stating behavioural objectives is indicated by Livingston (1980) when he states that

> the necessary starting point for (the process of instructional design) is a clearly defined instructional problem in the form of an unambiguously stated set of instructional objectives (p.21).

## TASK ANALYSIS

Once a main set of behavioural objectives has been prepared, it is often useful to closely examine each objective, precisely determining the steps the learner must take in order to attain that objective. The most expedient way of accomplishing this process is to perform what is known as a task analysis.

The process of task analysis involves the preparation of a hierarchical list (or chart) of the steps the learner must go through in order to meet the behavioural objective. When performing a task analysis, the instructional designer will normally examine the desired terminal behaviour and then work backwards, through "decending

levels of cognitive products and processes from abstract or general to concrete or simpler" (Hartman and Garnett, p.128).

When the task analysis has been completed, it is not uncommon to discover that the initial behavioural objective has now been broken down into subordinate objectives which are more readily measurable than the initial objective. Furthermore, particularly when the task analysis has been prepared in chart form, the existence of the various possible paths which learners can take to reach the terminal behaviour will become readily apparent. This discovery is especially helpful when designing CAI materials, since branching strategies can be designed which will allow the courseware to be more flexible in dealing with the needs of different learners.

## STRATEGY SELECTION

Once a task analysis has been completed, and behavioural objectives have been properly outlined, the design team must formulate the most appropriate instructional strategy for the chosen topic. This is an extremely important step in the instructional design sequence, and many decision factors must come into play when making this selection.

Many authors have categorized CAI into various instructional strategies. Although there is a certain amount of disagreement (ie. should games and simulations be considered to be two discrete strategies ?), CAI has typically been subdivided into the categories which are described on the following pages.

## Drill & Practice CAI

Once a new topic has been introduced to learners, it is often necessary for the learners to spend some time practicing the new ideas that have been introduced. This is particularly true when learning the new topic involves developing cognitive skills which occupy the lowest level (knowledge) in Bloom's taxonomy. Tasks such as learning the multiplication tables, memorizing poems, learning the basic arithmetic operations or remembering the order of letters in the alphabet all involve such lower level skills. Traditionally, the practice of such skills has involved the assignment of homework, the use of flash cards, in-class seat work etc. The transfer of such activites to the computer results in using the type of courseware commonly referred to as drill and practice.

Huntingdon (1979) defines a drill and practice program as, "a dialogue that repeatedly presents material until the responses meet criteria"(p.80). This rather broad definition clearly encompasses a wide variety of types of drill and practice programs, and several of these different types will be discussed in this section.

In spite of (or perhaps because of!) the fact that drill and practice was the most frequently used instructional strategy as recently as 1976 (Kearsley, 1976), drill and practice programs are "considered by some a mundane and perhaps inappropriate use of the computer" (Pogue, p.58). Other studies (Koch, 1973) have clearly indicated that, with an appropriate choice of topic (foreign languages and sciences in this case), "CAI drill and practice programs provide a significant advantage over traditional instruction" (p.29). While it is certainly true that some drill and practice programs prove to be ineffective, these programs are generally poorly designed, doing little more than repeatedly asking questions and providing "correct" or "incorrect" feedback to the learner. It is, of course, possible to design much more sophisticated drill and practice programs. Although this is often desirable, the reader should bear in mind that a more sophisticated approach requires more development time and that the, "complexity or sophistication of the drill or practice exercise (should) depend upon the difficulty level

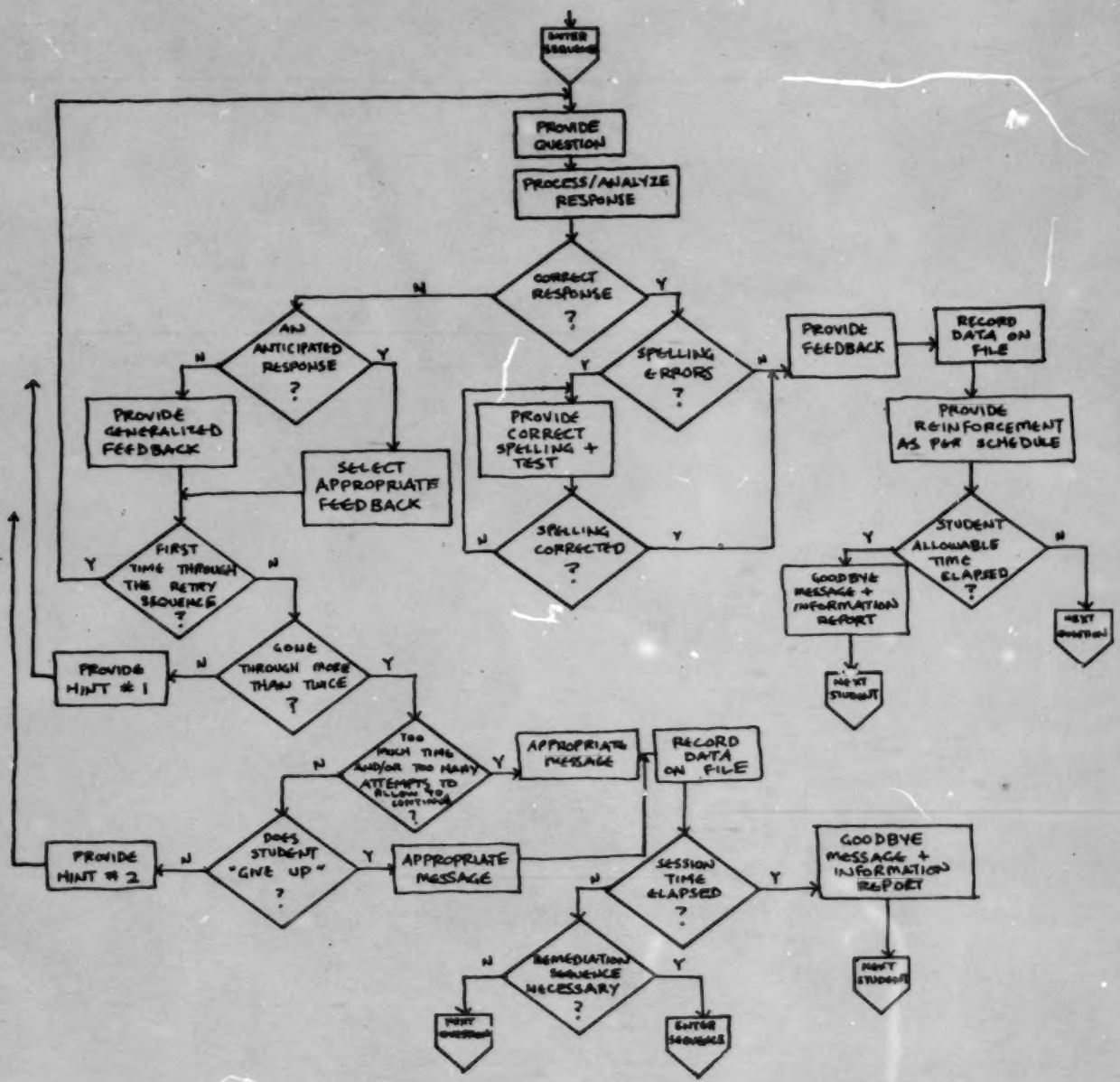
of the content and complexity of the processes in the specific type of learning setting" (Stewart, p.1).

A flow diagram for a question/answer sequence in a relatively complex drill and practice program is shown overleaf. The flow diagram falls naturally into various sections, each of which will be discussed separately.

Question:

Questions may be designed in many different ways. In drill and practice programs, questions are usually of the dichotomous choice (e.g. true/false) type, the multiple choice type, the fill in the blank type, or the matching type. When designing multiple choice questions, it is important that all of the distractors are equally plausible, that "there should be one clearly correct answer, and (that) all of the choices should match the stem grammatically" (Dennis, p.6). In many CAI programs, multiple choice questions are displayed just as they would be on a page of text, with all of the distractors being shown simultaneously. The computer, however, is capable of a different approach. One researcher (Bowles, 1978) used a technique in which the computer displayed only one distractor at a time, in a random sequence. The learner then indicated whether each choice was the correct or incorrect one. Of the strategies tested by the researcher,this strategy was considered to be, "one of the most effective" (p.152).

Questions may be generated in variety of different ways. Many arithmetic skills programs use the computer itself to randomly generate questions. By making alterations to the randomization section of the program, questions of varying difficulty can be generated. Another approach involves randomly selecting questions from a file or bank of designer generated questions. This is a more common approach when the questions are verbal in nature. A more sophisticated way of using randomization is to invoke a weighted random selection process. In this fashion, questions or question types that have previously been answered incorrectly are given an increased probability of being selected in future. Similarly, questions or question types that are repeatedly answered correctly may be given a


ENTER SEQUENCE
PROVIDE QUESTION
PROCESS/ANALYZE RESPONSE
CORRECT RESPONSE ?
N
Y
AN ANTICIPATED RESPONSE ?
N
Y
PROVIDE GENERALIZED FEEDBACK
SELECT APPROPRIATE FEEDBACK
SPELLING ERRORS ?
Y
N
PROVIDE CORRECT SPELLING + TEST
SPELLING CORRECTED ?
N
Y
PROVIDE FEEDBACK
RECORD DATA ON FILE
PROVIDE REINFORCEMENT AS PER SCHEDULE
STUDENT ALLOWABLE TIME ELAPSED ?
Y
N
GOODBYE MESSAGE + INFORMATION REPORT
NEXT STUDENT
NEXT QUESTION
FIRST TIME THROUGH THE RETRY SEQUENCE ?
Y
N
GONE THROUGH MORE THAN TWICE ?
N
Y
PROVIDE HINT # 1
TOO MUCH TIME AND/OR TOO MANY ATTEMPTS TO ALLOW TO CONTINUE ?
N
Y
APPROPRIATE MESSAGE
RECORD DATA ON FILE
DOES STUDENT "GIVE UP" ?
N
Y
PROVIDE HINT # 2
APPROPRIATE MESSAGE
SESSION TIME ELAPSED ?
N
Y
GOODBYE MESSAGE + INFORMATION REPORT
NEXT STUDENT
REMEDIATION SEQUENCE NECESSARY ?
N
Y
NEXT QUESTION
ENTER SEQUENCE

decreased probability of future selection. Drill and practice programs of this type are thus able to adapt to student needs. In some cases, it may be desirable to select questions from a file in an orderly rather than random fashion. This is particularly true when the subject matter is rigidly sequential in nature. When using an ordered selection, questions which have previously been answered incorrectly may be presented again according to a designer selected branching schedule. One common schedule is the 2,3,4 sequence. In this schedule, the item in question is presented again two questions after the initial attempt, five (2+3) questions after the initial attempt, and nine (2+3+4) questions after the initial attempt. Some types of ordered selection programs present an end review of "problem" questions at the end of the session.

Response:

When a learner has responded to a question, the computer must be able to analyse and interpret the response in order to provide the learner with meaningful feedback and/or reinforcement. Dichotomous choice and multiple choice questions require the least amount of analysis, whereas fill in the blank questions require considerably more evaluation. Dennis (1979) suggests that the computer should be able to perform the following analyses:

(i) recognize a correct response and all reasonably synonymous answers.

(ii) recognize literally incorrect, but conceptually correct answers.

(iii) recognize an array of frequently occurring incorrect answers.

(iv) recognize undetermined, probably incorrect answers.

Probably the most common type of analysis involves determining that an answer is correct, despite the presence of spelling errors. This type of response analysis can utilize an algorithm which detects frequently occurring

types of spelling errors (ie. letter reversal errors such as recieve/receive, freind/friend etc.). Other techniques might involve using a designer provided list of those incorrect spellings most likely to occur. Some computer languages and authoring systems have built in facilities for detecting spelling errors. Apple Pilot, for example, can automatically detect if a response is correctly spelled except for one incorrect letter. If spelling errors occur frequently, it may be desirable to include a brief spelling remediation sequence before allowing the student to proceed. However, it should be noted that such activities may distract the learner from the continuity of the learning sequence and thus actually interfere with learning. Furthermore, it may well be that the spelling "errors" are, in fact, artifacts resulting from the learner's unfamiliarity with the keyboard.

Feedback:

As mentioned elsewhere in this article, it is very important that learners be provided with informational feedback regarding their responses. In addition, it was suggested that the learning process was enhanced when the feedback was more detailed than a simple "correct/incorrect" message to the user. If the courseware has been designed in such a way that it is possible to detect anticipated but incorrect learner responses, then the learner can be provided with highly appropriate and useful feedback messages. If the learner ,for example, was asked to evaluate the expression "3 + 10 * 2 =___" and supplied an answer of 26, it is likely that the learner is forgetting to apply the rules of precedence in arithmetic operations. As such, it would be useful to provide the learner with an appropriate diagnostic message, rather than simply saying "no, that is incorrect...let's try again.".

It is generally desirable to include a response counting facility in a drill and practice program. This serves several purposes, perhaps the most useful of which being the correct timing of more general feedback messages to the learner. The instructional design team may prepare a series of "hints" which can be supplied to the learner after a pre-determined number of attempts at any given question. These hints will be particularly useful if

selected in conjunction with previous recognition of anticipated but incorrect responses. The response counting facility can also be used to allow frustrated learners to exit from a questioning sequence after a pre-determined number of unsuccessful attempts at a question. It is essential that learners not be caught in an endless loop in which they are continually presented with a question which they are unable to answer. The designer may wish to automatically exit the learner after a certain number of attempts or, alternately, may allow the student to respond to a "Do you wish to exit from this sequence ?" prompt.

Termination:

There are some situations in which learners, perhaps due to limitations imposed by the learner/computer ratio, can only spend a limited amount of time at the computer. If the hardware has some provision for recording elapsed time, it is helpful to have the computer itself determine when a learner's allowable time has elapsed. An appropriate termination message should be displayed at this time, perhaps providing the learner with information summarizing current performance results, or even suggesting remedial homework assignments (chosen from a bank of designer provided textbook listings). If an automatic timing facility is used, the learning sequence should be terminated at a natural exit point, and care should be taken to insure that a learning sequence is not abruptly terminated in mid-sentence!

Drill and practice programs often have built in classroom management capabilities in which information about the learner's performance is written to an external memory device upon termination of a learning sequence. This information may be relatively simple, recording information only about the number of correct and incorrect responses made by the learner. On the other hand, it may be useful to design a management system which records such parameters as response time, as well as all responses and spelling problems.

It should be clear that there is a great deal of flexibility in the definition and design of a drill and

practice program. Although generally considered the most simple category of CAI, a well designed drill and practice program can be very complex indeed. Many of the considerations outlined above (branching schedules, types of informational feedback etc.) hold true for the other CAI categories, and will not be discussed further.

Tutorial CAI

As mentioned above, drill and practice CAI is typically used to improve learner performance on topics that the learners have had some prior exposure to. Because drill and practice programs characteristically use rather limited types of questioning and tend to have rather restricted response analysis capabilities, they are not particularly well suited to introduce new topics of instruction. Tutorial CAI, on the other hand, is specifically designed to introduce learners to new information. Huntingdon (1979) describes a tutorial program as one which

> involves presentation of subject content, query for assesment of student learning, and re-presentation or alternate presentation when student learning does not meet criteria (p.107).

Although some authors (Peters and Johnson, 1978) take a different approach, in this paper the term tutorial CAI will only be applied to those programs in which questioning is computer directed. Programs which allow the learner to question the computer will be discussed under a separate heading.

The tutorial CAI category encompasses a wide range of program types. However, to varying degrees, tutorial programs generally attempt to mimic the broad flexibility and adaptiveness of the classroom teacher. Since the computer is being used to perform a task that, if performed by a human, would be considered a mark of intelligence, some researchers suggest that tutorial programs actually enter the realm of artificial intelligence (Dennis, 1979).

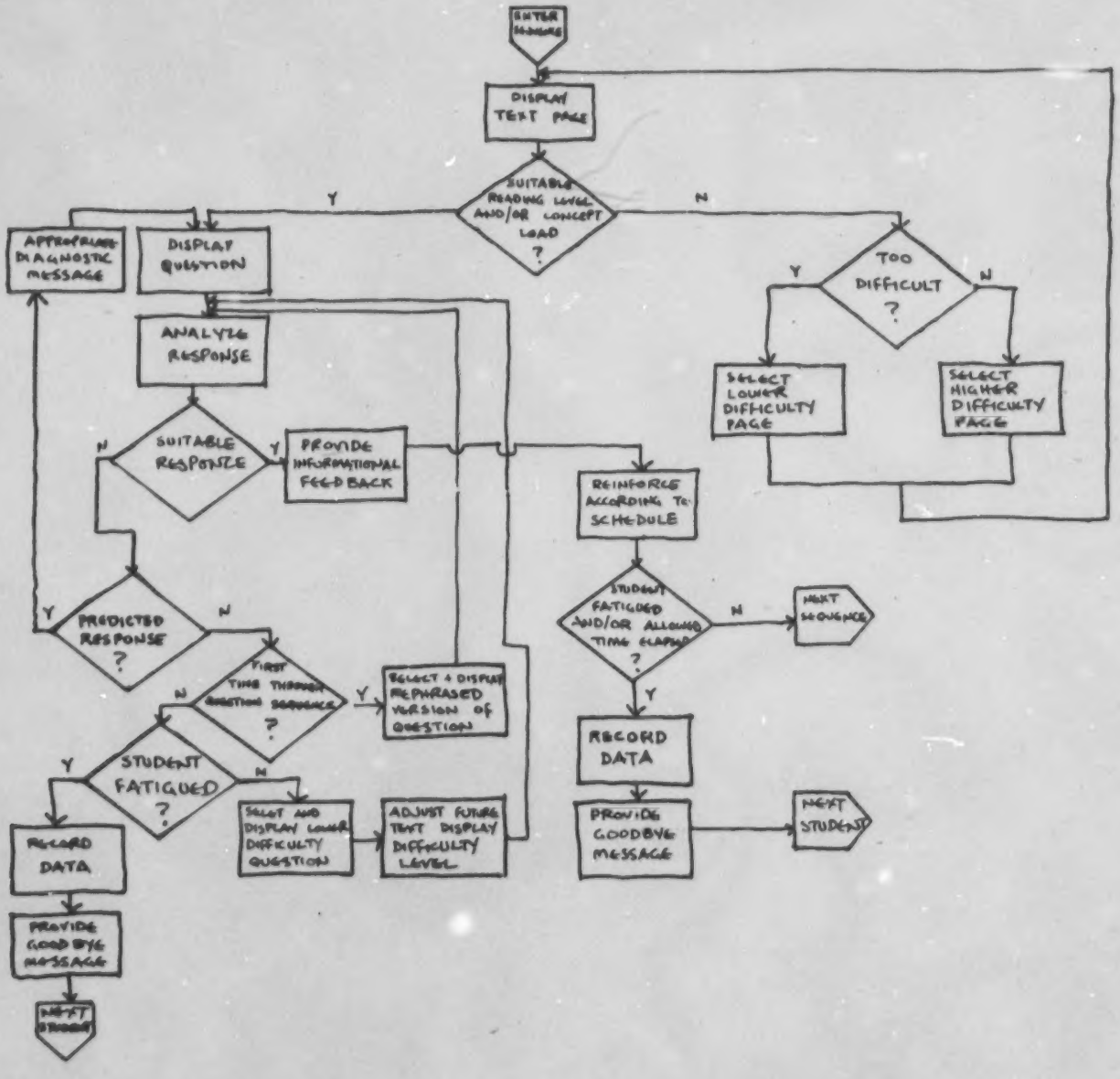
Due to the inherent flexibility of tutorial CAI, such programs tend to employ sophisticated, intricate branching schemes. Since these branching schemes are modelled on the flexibility of a real classroom teacher, it has been noted that "course authors with teaching experience can usually derive a richer repertoire of branching routines than those with little experience." (Huntingdon, p.108) The flow

diagram presented overleaf illustrates the adaptive nature of branching schemes in a fairly complex tutorial CAI program. Each main section of the flow diagram will be separately discussed below.

Text Page:

The initial textual information displayed to the student must be written in such a way that it is appropriate in both reading level and concept load for the grade level in question. However, students in any given class are rarely a homogenous group in terms of their ability to read and comprehend new information. Good classroom teachers are adept at detecting when learners are experiencing difficulty with the teacher's vocabulary level, and respond accordingly. In a similar fashion, a tutorial program should ideally be able to detect when the reading level and/or concept load of the text page is inappropriate, and make the appropriate adjustments. For example, when the learner initially begins to work on a tutorial package, the difficulty level of the text pages would be preset by the design team. However, the learner could be asked whether a more or less difficult text level was desired, and the computer could then select from a pre-written bank of alternate text pages. A more sophisticated approach would be one in which the learner, at any time, could alter the difficulty level of the textual information by adjusting the position of game paddle knobs or joystick handles. Furthermore, after several students had entered and worked with the tutorial program, the computer could "learn" that the initial difficulty level that had been selected by the design team was inappropriate, and make the corresponding adjustments.

Of course, highly sophisticated and adaptable programs will require more development time and, ultimately, a higher dollar cost for the end result. Equally important to consider is the fact that highly adaptive tutorial programs will require large internal and external memory allocations and that the complexity of the branching techniques will tend to slow down program execution time. While this may be of little consequence when working with mainframe or minicomputers, it may be that some microcomputers (at least the present models) are not suited


ENTER STUDENT
DISPLAY TEXT PAGE
SUITABLE READING LEVEL AND/OR CONCEPT LOAD ?
Y
N
APPROPRIATE DIAGNOSTIC MESSAGE
DISPLAY QUESTION
TOO DIFFICULT ?
Y
N
SELECT LOWER DIFFICULTY PAGE
SELECT HIGHER DIFFICULTY PAGE
ANALYZE RESPONSE
SUITABLE RESPONSE
N
Y
PROVIDE INFORMATIONAL FEEDBACK
REINFORCE ACCORDING TO SCHEDULE
PREDICTED RESPONSE ?
Y
N
STUDENT FATIGUED AND/OR ALLOWED TIME ELAPSED ?
N
NEXT SEQUENCE
FIRST TIME THROUGH QUESTION SEQUENCE ?
N
Y
SELECT & DISPLAY REPHRASED VERSION OF QUESTION
Y
RECORD DATA
STUDENT FATIGUED ?
Y
N
RECORD DATA
SELECT AND DISPLAY LOWER DIFFICULTY QUESTION
ADJUST FUTURE TEXT DISPLAY DIFFICULTY LEVEL
PROVIDE GOODBYE MESSAGE
NEXT STUDENT
PROVIDE GOOD BYE MESSAGE
NEXT STUDENT

to such sophisticated programs.

Question:

Unlike the questioning in drill and practice programs, questions posed in tutorial CAI tend to more closely parallel the types of questions a classroom teacher would use. Although the use of matching questions and cloze passages are sometimes used, questions requiring a short answer occur frequently. Such questions offer the learner much freedom to respond in a natural manner, however the learner's responses consequently require a considerable amount of computer analysis. In addition to the problem of spelling error detection described earlier, the computer must also be able to check for key words or portions of key words.

While the questioning in drill and practice programs tends to deal mainly with the knowledge cognitive level, questioning strategies in tutorial programs can adapt to explore the higher cognitive levels of Bloom's taxonomy. For example, once a learner had demonstrated proficiency at the knowledge level of a topic, the type of questioning could change, and focus on the comprehension level. This could, of course, then be followed by the next levels in the hierarchy until a given topic had been fully treated.

Response:

If a learner's answer is evaluated as being correct, informational feedback should be provided, followed by reinforcement according to a pre-determined reinforcement schedule. If vocabulary used by the learner when making the response is to be used in informational feedback, parsing algorithms may be incorporated so that the feedback uses appropriate grammatical structures. If the learner's response is judged to be an incorrect response of an anticipated nature, an appropriate diagnostic message should be given, followed by a repeated presentation of the question. If, however, the incorrect reponse is of a type that has not been anticipated by the design team, the user should be provided with some general informational feedback and then provided with a rephrased version of the same

question at the same difficulty level. In this way the computer can simulate a classroom teacher and adjust phrasing to meet the learner's needs. If the learner still cannot answer the question successfully, it may be that fatigue is the problem. As discussed in the learning theories chapter, duration of the learning session plays an important role in the efficiency of the learning process. Learners should be asked if they would prefer to stop now and return at a later time. If fatigue does not appear to be the problem, the computer should automatically alter the difficulty level of the question and, at the same time, adjust the future text display difficulty level. It may be that, perhaps due to high motivation, the learner previously selected a difficulty level that was too high, and was therefore experiencing difficulty.

Termination:

As with drill and practice programs, a mechanism must exist by which students are exited from the learning sequence. This may be achieved automatically, or the learner may simply request an exit at the appropriate time. When learner data is written to an external memory device, information such as learner placement in the tutorial sequence, as well as both question and text display difficulty level should be recorded.

In summary, tutorial CAI can obviously be extremely complex, particularly when programs are designed to be highly adaptive. Just as some computer chess programs employ heuristic problem solving techniques to gradually evaluate and "learn" the most effective move sequences, highly sophisticated tutorial programs may use similar strategies to gradually evaluate and "learn" the most effective reinforcement schedules, question types, question strategies and initial levels of reading difficulty.

Dialogue CAI

As outlined above, tutorial CAI consists primarily of a series of text display, question and answer episodes and is essentially computer directed. In contrast, dialogue CAI, while still retaining the elements of text display, questions and answers, is learner directed. In this way, it is the learner, rather than the computer that poses the questions. Dialogue CAI, to an even greater extent than tutorial CAI, is an attempt to simulate the flexibility and adaptive capabilities of a classroom teacher.

The most difficult task involved in designing learner directed courseware involves the analysis of, and appropriate response to , learner queries. Information processing techniques must be invoked if learners are to be allowed to pose questions of any length. Due to the inherent ambiguity of the English language, obvious problems of interpretation will arise. For example, the word "induction" can take on various meanings, depending on the context in which it is used. This difficult problem can, however, be reduced somewhat by limiting the learner to a rigid sphere of context (microworld) in which words only have one meaning. For example, in a dialogue program designed to introduce learners to the historical concepts of conflict, learners could be instructed to restrict their vocabulary to the microworld of sociology. In that context, "induction" would refer to military service and have nothing to do with the production of an electric current.

In dialogue courseware, the computer will generally scan a learner's question, searching for a key question word (who, what, where, when etc.) and a key subject word. Often, the computer will also be required to search for the root of a key subject word and respond in an appropriate grammatical context. For example if a learner asked "What does it mean to be inducted ?", the computer would scan the query and determine that the key question word was "what" and that the key subject root was "induct". The learner could then be provided with a grammatically correct response such as "Induction is the process by which...".

A good dialogue program could present information in a

standard tutorial fashion, asking the learner questions, and analyzing responses. However when the learner encountered difficulties, the program could switch to dialogue mode and help the learner to make relevant queries to the system.

Dialogue programs that are capable of responding to complex learner queries require both extended development time and skillful programming. In addition, programs of that nature tend to require large memory capabilities. Although it is feasible to implement such a program on a microcomputer, the costs may be prohibitive. In addition, sophisticated dialogue programs are probably best written in one of the list processing languages, and these are not readily available for microcomputers. Some simpler programs might restrict the learner to queries that were made up of only a single key word, and the learner would then be provided with a simple definition.

In summary, it should be noted that dialogue CAI requires much development and programming time. In fact one author (Huntingdon, 1979) suggests that dialogue programs

> are the most difficult types of CAI to produce and the returns on the investment in producing them should be carefully projected. (p.109)

## Simulation CAI

Simulation programs can typically be described as serving as a "replication of the behavior of a phenomenon of one's universe designed to substitute for the phenomenon" (Dennis, p.2). A simpler definition describes simulations as an attempt to "authentically model reality" (Shay, p.26).

Unlike the types of CAI discussed above, simulations are generally not used as a device for providing learners with factual information. Shay (1980) has established a list of objectives which classroom simulations are typically designed to accomplish. Although Shay was specifically discussing non-computerized classroom simulations, many items in the list are also appropriate to simulation CAI.

### To Develop Empathic Sensitivity

Simulations which fall into this category would often involve role-playing. By taking on the role of someone in another culture or situation, a learner could begin to develop a sensitivity to the problems and circumstances that other person might experience. For example, taking on the role of an early explorer would enable the learner to become aware of the difficulties of such a life.

### To Examine the Dimensions and Dynamics of Social Problems

Simulations like this also involve role playing to a certain extent but, unlike the simulations above, modern day, politically and socially relevant situations are explored. Such simulations are designed to introduce learners to the complexities and ramifications of modern social and political issues. For example, a simulation program might illustrate the dangerous situations created during a hostage taking incident.

To Explore the Future

Some simulations are designed to allow learners to extend real life situations to a number of possible futures and, in doing so, acquire a heightened awareness of the implications of actions taken in the present. The well known global future simulation programs are examples of simulations which fall into this category.

Whereas the simulation categories listed above primarily deal with social issues, simulation CAI is also used extensively in scientific situations. Generally speaking there are five instances in which computer simulations in the science classroom prove most useful. These are situations in which:

(i) The necessary equipment and supplies would be too costly to purchase.

(ii) Performing the real experiment would expose the participants to possible danger.

(iii) The laboratory techniques required to carry out the real experiment would be beyond the capabilities of the learners involved.

(iv) The duration of the real experiment would be either too long or too short to be suitable for a science classroom.

(v) It would simply not be possible to carry out the real experiment with existing technology.

Simulation CAI can clearly be employed in many situations where other types of CAI programs would have limited use. Furthermore, research indicates that learners very much enjoy simulations, and are quite eager to spend a great deal of time using such programs. (Shay, 1980; Dennis, 1979) Their very popularity can, however, cause problems with classroom management, particularly when computer resources are limited. Furthermore, the development of simulation CAI is characteristically a

difficult, time consuming and costly process. Courseware designers should be cautioned that, because of the inherent unpredictability of simulations, simulation programs should be thoroughly tested under a wide variety of situations. In addition, due to the unpredictable outcome of simulation programs, teacher directed follow-up discussions are recommended. Furthermore, of course, it is essential that the mathematical model used in constructing the simulation be an accurate representation of the real world.

Games CAI

Whereas some simulation CAI programs are designed to be games, there exists a category of game programs which do not model the real world. Programs of this type will be discussed in this section.

Most authorities generally consider games to have the following common characteristics:

(i) A number of participants, usually, but not necessarily, more than one. In the case of CAI, the role of one or more of the participants (players) may be taken by the computer.

(ii) A circumstance recognizable by the player(s) as denoting the end of the game. This circumstance may be defined at the beginning of the game, in which case it is called a goal.

(iii) A number of rules outlining the way in which the game will be played. Such rules are usually non-ambiguous, although the goal of some games is for the player(s) to discover a set of undeclared, ambiguous rules. Some researchers (Shay,1980) further categorize rules as:

a) Procedural Rules
b) Behavioural Rules
c) Resource Allocation Rules
d) Evaluative Rules
e) Mediation Rules

(iv) A means by which players may determine the state of the game at any time.

Most "pure" games (ie. games that do not attempt to overtly simulate the real world) are found in mathematics. Nim, the Towers of Hanoi, the MU game (Hofstadter, 1980) and various cypher games are common examples.

In many cases, teachers and administrators find it difficult to justify the use of games in the classroom.

While it is certainly true that some games have no redeeming educational value, there are other instances in which games, particularly mathematical games, have been shown to encourage learning (Dennis et al., 1979).

While it appears reasonable to suggest that games are useful as a means to increase learner motivation, and perhaps develop analytical reasoning skills, it is still important to bear in mind that

> when games are used there must be a thoughtful plan that includes organizing activities preceding the game, and summarizing and relating activites folowing the game (Dennis et al., p.13).

In summary, the reader should note that, in courseware design, the categories of CAI described above need not be considered as discrete entities. In fact, good CAI often utilizes several instructional strategies within one sequence of programs. Furthemore, since the ultimate choice of instructional strategy will exert a great influence on the ease with which student learning will be facilitated, it is essential that the instructional strategy be made appropriate to the learning task.

## PREPARE A STORYBOARD

At this point, decisions will have been made regarding the goals and objectives of the courseware, as well as the optimum instructional strategy or, more likely, combination of instructional strategies which will be implemented. It is now necessary to provide the programmers with a detailed description of exactly what will appear on the video screen at all times, as well as a detailed plan of exactly what information the program must record (ie. number of attempts at a question, reading level selected etc.)

A very efficient way of providing the programmers with this information is to prepare a storyboard. This may conveniently be accomplished in four steps:

(a) Prepare a simplified flowchart, outlining all the different paths a learner might take to move through the program. Details of the information which the learner will see need not be included in the flowchart. The flowcharts appearing on pages 25 and 32 are examples of the amount of detail necessary. The purpose of preparing the flowchart is merely to conceptualize just how the computer will control a learner's progress through the program.

(b) Divide the flowchart into smaller, easily handled segments. The segments should begin and end at natural "critical points" in the program. For example, there may be an introductory segment, five core instruction segments, two remedial instruction segments, three enriched instruction segments, a report generating segment and a spelling error detection segment.

(c) Take each segment and expand it into a somewhat more detailed flowchart. A flowchart box which simply stated "display text sequence" in the steps above could now become four boxes stating:

-introduce new vocabulary
-display diagram
-describe diagram

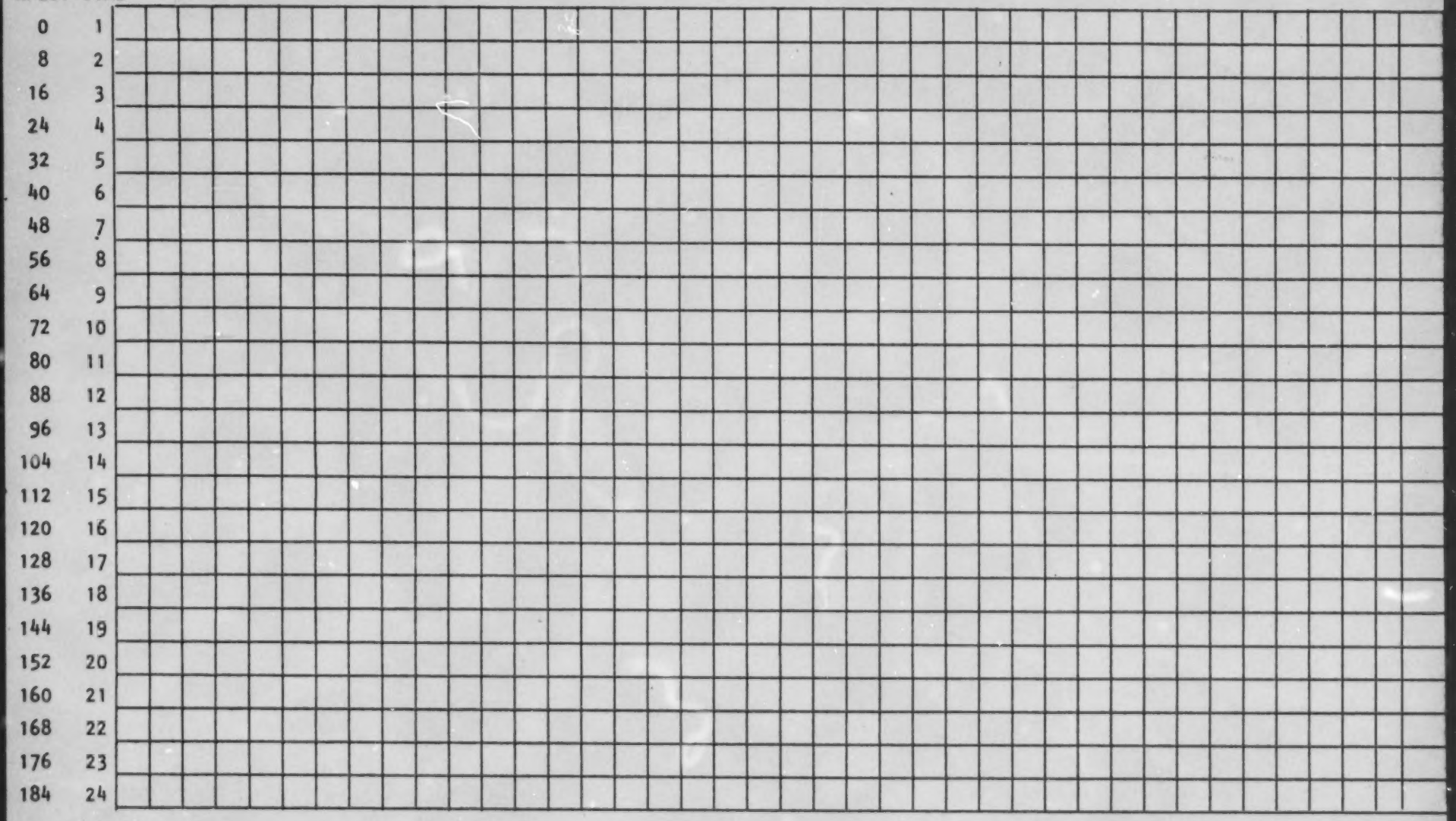
-present instructions

Each of the new boxes should be clearly labelled, indicating both the number of the box, and the segment in which the box appears. A convenient method is to label each box with a letter and a number. The label "F7" would then indicate that the box appears as box 7 in segment F.

(d) Finally, prepare a screen layout sheet for each of the labelled boxes. The layout sheets should illustrate exactly what must appear on the screen at that time. No ambiguities of spelling, punctuation, or placement within the screen should be present. Be sure that the amount of text that has been specified for each screen layout sheet matches, in both amount and format, the amount of text the hardware can display. For example, do not specify an 80 column text line of upper and lower case letters when the computer is only capable of displaying 40 columns of upper case letters. The preparation of screen layout sheets is facilitated if graph paper which corresponds to the text display capabilities of the hardware is available. For example, graph paper using a 40 by 24 grid, as shown overleaf, would prove useful if the chosen computer was an Apple II microcomputer.

When designing the screen layout sheets, it is of utmost importance to use the available screen space in an appropriate manner. The design team should pay particular attention to the manner in which textual information is displayed on the screen, the way in which the system will accept learner input, and to the way in which the learner is provided with directions for using the courseware.

Since text can rarely be presented on a video screen with the same resolution that the printed page offers, several factors must be taken into consideration in order to maximize legibility. If possible, text should be displayed using both upper and lower case letters, and the text font should be of a standard design. The size of the text should be appropriate to the reading ability of the learner, but large letters consisting of low resolution 'blocks' should be used with caution, as they tend to have




PROGRAM _____ SCREEN _____

limited legibility. Text, particularly when the learners are quite young, should be double spaced and centred on the screen. Words that are to be emphasized should be underlined or displayed in a different font (bold face for example). Inverse or flashing text should be avoided, since poorly tuned video displays cannot represent such text with any degree of clarity. Care should be exercised in order to insure that the text is displayed one frame at a time, as opposed to the less readable scrolling text display. The amount of information (concept load) appearing in each frame should not be excessive, and the frames should never terminate in mid - sentence, particularly when it is not possible for the learner to 'step backwards' and review earlier frames at will. The text should appear on the screen in a consistent fashion, with recurring events such as questions or explanations appearing at the same place on the screen every time. Learners should never be presented with a blank screen for any length of time, and if the design team suspects that a particular computation or disk access may involve a lengthy delay (more than a few seconds), an explanatory message should appear on the screen. For example, if the program must retrieve a graphic display from the disk, the screen should inform the learner with a message such as 'Please be patient while your answer file is loaded'. The text should strictly adhere to the correct rules of grammar, spelling, punctuation and hyphenation. The reading level of the text should be appropriate to the ability of the prospective learners, and several readability formulae are available for estimating reading difficulty. However, the design team should note that, particularly with video displays, the legibility of the text and the concept load per frame are factors which may considerably alter the reading difficulty, as calculated by one of the traditional formulae.

The design of the courseware should be such that instructions to the learner are kept at a minimum, as the repeated appearance of user instructions tends to draw attention away from the desired focus of learning. Readers should consider the principles of virtuality in interactive systems (Nelson, 1980), in which it is suggested that the reality of a computer program (its data structures, branching schemes, programming language etc.) be invisible to the user, and that, "an interactive computer system for

most purposes should be learnable in under ten minutes" (p.57). With this in mind, learners should never be forced to type syntactically rigid commands like 'RUN INSTRUCTIONS', but should instead be allowed to respond to interactive system queries. Many courseware authors use a "menu" approach to allow learners to select programs to run. When the learner must be provided with instructions, the instructions must be clear and unambiguous. The learner should have the ability to control the presentation of frames, both forwards and backwards, and the instructions for doing so should be consistent in both content and screen location. Furthermore, learners who are familiar with the courseware instructions should be able to suppress their appearance on the screen. Learners should have the ability, at any point in the program, to request help and then be presented with a 'help frame'. In addition to the more obvious instructions that should appear on the help frame, the learner, perhaps tired or frustrated, should be given the option of terminating the session.

If a CAI program is to be considered to possess the qualities of virtuality, it is absolutely essential that learners are not able to halt execution of the program by accidentally generating a system run-time error. Since such errors are most often generated when the learner inputs inappropriate information in response to a system query, user responses must be carefully analysed prior to being formally accepted by the program. Inappropriate responses may be handled in various ways, such as:

Ignore and Proceed:

In this situation, when the learner supplies the program with inappropriate information, the program simply ignores the response and proceeds. Common examples of this type of error handling occur when learners type the word 'three' instead of the number '3' in response to a question. The program, using an error trapping routine to convert the word into the corresponding number, is then able to ignore the fact that the learner's response was incorrect, and proceed. Similar routines can be designed to detect obscenities or ridiculous responses when learners are asked to input their name. When the program detects

such a response, the lesson proceeds, referring to the learner simply as 'friend' from that point on.

Check and Repeat:

When it is not possible for a program to proceed following an incorrect learner response, it is sometimes appropriate to simply repeat the question. For example, if a learner is prompted to press the space bar to proceed to the next frame, but instead presses the return key, the prompt can simply be repeated.

Check, Assist and Repeat:

There are situations in which learners who have made an inappropriate response should be provided with some assistance before being prompted for another response. For example, if a learner responds to a system prompt by suggesting zero as a divisor in some arithmetical expression, the learner should gently be informed that division by zero is not allowed. If the question is simply repeated without any explanation or assistance, the learner is likely to become confused and frustrated, and a situation in which program execution is terminated with a '***DIVISION BY ZERO IN LINE 11435***' error message would be disastrous.

Full details of all such input processing procedures should appear on the final screen layout sheets as they are provided to the programming team.

## EXAMINE THE MATCH BETWEEN DESIGN & IMPLEMENTATION

The final step prior to the beginning of the coding sequence must involve the programming team. The programmers will examine the storyboard, and determine if there are aspects of the courseware design which will not be possible to implement. It may be that the hardware is inadequate in terms of memory capacity, graphics capability etc., or it may be that the language that the programmers have opted to use is not capable of faithfully executing the program design as it appears on the storyboard. At any rate, if problems arise from an inadequate match between design and implementation, some revisions must obviously be made before the coding sequence can begin.

## CODING SEQUENCE

When a storyboard and a set of screen layout sheets have been prepared, and when the programming team agrees that such a design can be faithfully implemented using the chosen hardware and software combination, then the design sequence of the courseware development scheme may be considered to be completed.

Although individual style and preference may vary, most authorities now feel that the coding sequence should proceed according to the principles of top-down program design (refer to Appendix I for a description of top-down programming).

It is important that full annotation of the written code takes place during the coding sequence. A detailed description of this process is described in the following chapter.

If the chosen language supports the use of local variables (ie. like APL and Pascal), the programming team should make an effort to make all variables used in a program module local to that module. In this way, program modules can be fully tested and debugged prior to connection with the main program. Furthermore, when local variables are used, the resulting program modules become "stand-alone" subroutines and can, without change, be safely used in other programs, time and time again.

Once all of the modules which will be used in a program have been written and fully debugged, the main program can be assembled and tested. Generally speaking, if program design has proceeded according to the rules of structured programming, little debugging of the fully assembled main program should be necessary.

At this point, when the entire program is fully operational, it is useful to let the design team view the running program. The design team can then determine if further refinements to the program are necessary. When refinements are required, it is generally because the program has not been fully "bullet-proofed", or because

certain portions of the program do not execute quickly enough.

Bullet-Proofing

As described in a previous chapter, it is absolutely essential that a learner not be able to accidentally (or even intentionally!) halt the execution of a CAI program. The process by which error checking routines guard against this process is often referred to as "bullet-proofing" the program (some less diplomatic program designers refer to this process as "idiot-proofing" the program!). One of the best ways to determine if a program is fully bullet-proofed is to execute the program many times, each time deliberately trying to halt program execution. The program should be able to handle all of the following "errors" satisfactorily.

(a) Division by zero errors caused by the user entering a zero when requested for information should not cause a program to crash.

(b) Pressing "Control-C" (the escape key on some computers) should not cause program stoppage.

(c) Pressing the reset key should not cause the system to halt. This key should either be disabled completely, or else should cause the system to reboot.

(d) Leading and trailing blanks in user input should not cause any problems. For example, when matching learner responses with a data list of correct responses, the presence of leading or trailing blanks in an otherwise correct users response should never cause the system to treat the response as incorrect.

(e) Incorrect types of learner responses should not cause program disruptions. For example, if a learner types 'three' instead of '3', no problems should arise.

(f) If the computer supports both upper and lower case characters, the mixing of upper and lower case in a learners response should not cause difficulties. For example the response 'three' should be handled in the

same way as 'Three'.

(g) Very long learner input strings should be processed without difficulty.

(h) Illegal quantities should not cause a program to stop execution. For example, if the maximum integer size allowed by the system is 32 767, a learner input of 55 000 should be handled appropriately.

(i) Use of the backspace, delete or rubout keys should allow the learner to correct input prior to pressing the return key.

(j) The keyboard strobe should be cleared after every learner input. In this manner, pressing the return key (or spacebar) should not cause premature presentation of the next frame.

(k) Obviously absurd learner responses should not be accepted.

(l) Profanity checking routines should be implemented prior to accepting learner input.

(m) When material is to be written to a disk during program execution, the program should guard against such errors as a write-protected disk, a disk with insufficient free memory space to store the required information, or a completely full disk. The learner should be given the opportunity to either remove the write-protect tab, or insert a new disk.

(n) When material will be read from a disk during program execution, the program should guard against such errors as the disk drive door being accidentally left open, or the wrong disk being left in the drive. The learner should have the opportunity of either closing the drive door, or inserting the appropriate disk.

## Increasing Execution Speed

Although this is ordinarily not a problem, occasional situations arise in which a portion of a program is executing too slowly. This sometimes happens when a particularly complex computation must be tackled, or when a particular animation sequence is found to "flicker" too much.

Unfortunately, most efforts to increase program execution speed tend to make a program less readable. Therefore such program alterations should only occur when absolutely necessary.

The following modifications will increase execution speed to a certain extent.

(a) When a series of IF statements are used, put the condition which is most likely to occur at the head of the list. In this way, the computer will be able to skip the rest of the IF statements most of the time.

(b) Use of multiple statements per line.

(c) Place frequently referenced lines at the beginning of BASIC programs.

(d) Use variables rather than numerical constants in BASIC programs. Fortunately, when meaningful variable names are used, this also tends to increase program readability.

(e) Omit the index variable in FOR NEXT loops in BASIC (ie. simply use NEXT instead of NEXT J). However, be very cautious about this facility, as tnis seriously detracts from program readability.

When a program has gone through these modifications to make it bullet-proofed and (if necessary) speed-efficient, the design team should then examine the running program once more. If the program is deemed satisfactory, the coding sequence is considered to be tentatively complete.

## DOCUMENTATION AND EVALUATION SEQUENCE

The writing of a courseware package cannot be considered complete until the courseware is fully and effectively documented. Furthermore, the courseware development process cannot be considered complete until this fully documented package has been fully and objectively evaluated.

It is, unfortunately, still not uncommon to discover that many of the currently available courseware packages are marketed without adequate documentation of any kind. Undocumented or inadequately documented courseware packages are a source of endless frustration to classroom teachers who have neither the time nor the inclination to spend hours trying to determine how to successfully use the courseware. It is indeed unfortunate that much, otherwise excellent, CAI is simply left unused because it is poorly documented and therefore too difficult to get "up and running".

The documentation process consists of two distinct tasks. The program itself must be documented and, in addition, there must be fully detailed support materials available to accompany the program. The two types of documentation tasks will now be discussed in detail.

### Documenting Code

A program can be considered well documented if it is replete with liberal programmer comments and explanations. Programmer comments should be embedded within the code at the time the code is written. In this way, programming ideas are fresh in the programmers mind and, thus, more clearly explained. The importance of documenting the code during the coding sequence is stressed by Magel (1981) who suggests that

> More time is spent trying to figure out what a section of code is supposed to do and how it is

> trying to do it than would be spent documenting that information as the code was written...the longer one waits after the code is written to document it, the more difficult it is to remember accurately what was done. If someone other than the programmer prepares the documentation, the necessary imperfect communication between programmer and documenter greatly lessens the chance that the documentation will be accurate (p.147).

When documenting code, it is important that programmers do not omit descriptions of program details with which they feel the eventual users will "probably" already be familiar. Robinson (1980) feels that

> Possibly the greatest problem in writing effective documentation is the fact that the programmer who created the program often finds himself assuming that the reader knows a great deal more than he actually does. The programmer may find it difficult to explain the program in terms that someone else can easily understand. Being the author, and thus knowing well the background and details of the program's usage and operation, he tends to omit those details from his documentation and assume that the reader already knows and understands them. In fact, he probably does not (p.30).

All variables, constants, functions, procedures, and other such features should be fully described by programmer comments. This process is facilitated when "self-documenting" languages such as Pascal are used, but a clear identification of data structures should be present regardless of the language used. In fact, even if a "self-documenting" language is used, additional programmer comments are always necessary to fully document the code.

The code should be designed for maximum readability and, if possible, programmer comments should be written in such a way that they are clearly distinguished from the executable code.

The following identification comments should be present at the beginning of the program:

(a) Program title.

(b) Programmer's name.

(c) Date that original coding was completed.

(d) The current version of the program.

(e) The date that coding of the current version was completed.

(f) Peripheral devices necessary to use the program.

(g) Memory requirements of the program.

(h) Name of the disk operating system used.

(i) Version of the programming language used, including deviations from ANSI standards.

It sometimes occurs that the liberal inclusion of programmer comments will cause a program to occupy too much memory space at run time, with the consequent generation of run-time "out of memory" errors. If this is the case, two versions of the program may be supplied. The version designed to be executed contains no programmer comments, whereas the fully documented version is designed for listing only.

External Documentation

External documentation should consist of a catalogue-like summary of the various features of the courseware, a copyright notice (if required), a developer's rationale, statement of goals and objectives, an instructor's guide and (if necessary) a learner's guide. Unlike the documentation of the code, the external documentation should be prepared by the instructional

designers.

The summary of courseware features should be designed in such a way that prospective users of the program can quickly and easily determine if the materials are suited to their needs. The summary should consist of the following:

(a) The title of the courseware package.

(b) A brief description of what the package does (an abstract).

(c) The grade level at which the package is designed to be used.

(d) The readability level of all learner-oriented text.

(e) The programming language used.

(f) Memory requirements.

(g) Peripheral requirements.

(h) Disk operating system required.

(i) Approximate running time of the programs.

The developer's rationale should explain the developer's reasons for producing the courseware materials. The particular learning theories accepted by the development team, and the position of the development team on certain social issues (i.e. computer literacy issues) are factors often cited in the rationale statement. In addition, the rationale statement may reflect the developer's response to governmental policies on curriculum.

The developer's rationale statement serves as a foundation for statements of goals and objectives. As described in the design sequence chapter, goals are broad, general statements outlining the expected outcomes resulting from learner use of the materials. They are stated so broadly that teachers could not use them as a

basis for daily lesson planning. Objectives, on the other hand, are much more specific statements and are often stated in terms of observable behaviour.

An instructor's guide should provide prospective instructors with complete directions for using the courseware package. Full descriptions of the following items should be outlined in the guide:

(a) Any specific entry competencies required for use of the materials.

(b) Any suggested teaching methodologies, along with descriptions of suggested classroom activites or "typical lessons".

(c) Any preperations an instructor will have to make in order to successfully use the courseware package.

(d) Any of the ways in which students who use the materials will be evaluated. This is particularly important when the courseware package has sophisticated built-in classroom management capabilities.

(e) Any particular technical considerations, such as disk loading procedures, print-out procedures, disk-copying capabilities etc.

(f) Any general peculiarities of which learners must be made aware (i.e. use of the return key, spacebar etc.).

(g) Any additional sources of reference material.

(h) Any significant results from field-testing of the materials.

In general, the instructor's guide should be designed to allow instructors to feel at ease with the courseware package, and experience as few problems as possible.

The learner's guide should attempt to maximize the ease with which the learner will interact with the courseware materials. Specific behavioural objectives

should be stated prior to each logical division in the guide and, in addition, the guide should describe any specific input formats, should explain how to request help or printouts, and should generally outline how the learner can proceed through the materials.

When the courseware package consists of fully documented, error-free and bullet-proofed code, along with complete user and instructor documentation, then it may be considered ready for field testing and evaluation.

## Field Testing and Evaluation

Courseware should always be field tested prior to final release. Gerhold (1979) stresses the fact that "evaluation of courseware for the purposes of editing and improvement of that particular course is mandatory" (p.21).

Examination of the formal, objective evaluations which result from the field testing procedures is the most efficient way in which the courseware design team can determine if the instructional package actually achieves the broad goals initially defined in the design sequence.

Careful analysis of formal courseware evaluations can help the design team discover what features in the package were particularly effective and which features were ineffective. It may be necessary, after examining the results of a courseware evaluation, to make revisions in the package. As Magel (1981) says

> Very few, if any, projects go from start to finish in smooth progression. Even the best planning must leave room for the unexpected to occur. When it does, the project must be prepared to move back to an earlier development phase and continue from there (p.152).

The nature of the changes which take place depends, of course, on the gravity of the courseware flaws which have surfaced as a result of close examination of courseware evaluations. It is sometimes only necessary to make minor alterations to code, and this should proceed with little difficulty if the code has been well documented. At other times, however, more serious alterations within the planning or design sequence must be made. Sometimes, radical alterations are called for, and in extreme cases, it is best just to start over from the beginning, because "At some point, then, writing a new course makes greater sense than trying to figure out how to change the old one" (Mager, p.152).

The accuracy with which a courseware evaluation reflects the true quality of the courseware depends, at least partially, on the quality and effectiveness of the

evaluation instrument itself. A number of courseware evaluation instruments exist, and the choice of an instrument depends, to a large extent, on the complexity and cost of the courseware package which is to be evaluated, and on the purpose for which the evaluation is being made. Two evaluation instruments are included in Appendix III.

The "School Microware Evaluation Form" is a checklist-type evaluation instrument that should take from 30 minutes to an hour to complete. The form is extremely useful for the reporting of gross flaws in courseware materials, but has no provision for the reporting of more subtle problems such as social considerations or curriculum match.

The "JEM Research Microcomputer Courseware Materials Analysis Form" is considerably more detailed, and enables the evaluator to report on many of the more subtle features of the courseware. Although the instrument may be used by anyone, prospective evaluators may, at least initially, experience some difficulty with the somewhat specialized vocabulary used in the instrument. As such, the instrument is probably used most efficiently by qualified analysts of educational materials.

The courseware design sequences described on the previous pages, if followed, should result in the production of quality software in an efficient, cost-effective manner. Good luck!

# APPENDIX I: CHOOSING A LANGUAGE

## LANGUAGE CHOICE

A bewildering array of computer languages exists today, each of which has its own unique vocabulary of statements and commands. Some computer languages are designed to accomplish tasks of a specific nature, whereas others are less specific in nature and thus can accomplish many tasks. Of this plethora of languages, as yet only a subset has been configured to run on microcomputers. Since the readers of this publication are likely to be designing courseware which will ultimately run on a microcomputer, the description of computer languages which follows will be restricted to those languages which run on microcomputers, either presently, or in the near future.

The choice of an appropriate language for a CAI program is an extremely important one, and one author (Atherton, 1979) states that the "nature of the computer language which schools use most widely will clearly influence or inhibit all kinds of developments" (p.205). Many factors will influence the ultimate decision about which language to choose. Some of the most important factors which must be considered are:

(a) portability of the language
(b) support of structured programming techniques
(c) execution speed
(d) purpose of the language
(e) ease of learning and programming

The emphasis which will eventually be placed on each of these factors will, of course, depend to a large degree on who will be expected to carry out the actual programming task. If the courseware design project is a major one, an external programming team will likely be hired to do the programming. For example, the CONDUIT institution at the University of Iowa employs externally hired professional programmers for 50% of their courseware development (Peters and Johnson, 1978). When professionals are hired, the choice of a programming language will probably be left largely up to the programmers themselves. The courseware design team, after presenting the programmers with a fully detailed program storyboard, might only express concern about the ultimate portability and adaptability of the

resulting software.

However, if teachers or students will be doing the actual programming, how easy the language is to program in, and to what degree the language supports the concepts of structured programming will probably be the most important factors affecting the choice of a language.

Each of the factors mentioned above will now be described, and a brief overview of some of the common microcomputer programming languages will follow.

## Portability

A program or language is generally considered to be fully portable if it can execute, without change, on different types of computers. If a CAI program is designed only to be used locally, on one particular brand or type of microcomputer, then the problem of portability need not be considered. However, courseware is often written with the idea in mind that it will eventually receive wide distribution, running on many different types of computers. In cases like this, an effort should be made to select a language that is as machine independent as possible. Furthermore, once such a language is selected, the programming team should refer to the most recent ANSI standards publications for the chosen language, and refrain from using any non-standard ANSI features in the program.

## Language Purpose

It is important that the programming language which is selected is one which is suited to the particular programming task at hand. As suggested by Lower (1981), some languages may

> simply lack the needed facilities, such as string handling or array operations. More frequently the language is fundamentally doing the job, but at the cost of awkwardness or even drudgery in programming, which inevitably degrades the quality of the final application program. (P.21)

For example, a language able to quickly carry out complex mathematical or scientific computations (ie. FORTRAN) is an unlikely candidate for a CAI program which is largely text oriented. In a similar fashion, a language with relatively strong text processing capabilities (ie. PILOT) will be ill-suited to CAI programs which involve a fair amount of number crunching.

Execution Speed

Computers cannot directly execute instructions which have been written in a high level computer language. They are capable of directly acting upon instructions only when the instructions are presented in machine readable form. There are two fundamental ways of transforming a program into machine readable form. The program can be either interpreted or compiled.

A compiler will take an entire program written in a high level language and translate it into a machine readable form. This step need only be performed once, and the program is henceforth stored and executed in this machine language form.

Interpreters, however, translate the high level program one line at a time, avoiding the compilation step and thus executing the program immediately. However, the program is stored in its original source form and must be reinterpreted every time it is executed.

As a general rule, compiled code executes at least ten times faster than code which must be interpreted. In addition, the compiled code takes up much less memory space than uncompiled source code. Programs which require high speed animation sequences, complex calulations or other such features which demand a high program execution speed are probably best written in a compiler based language.

Ease of Programming

Teachers or student programmers who are relatively new to computing may wish to select a language which is both easy to learn and, once learned, is easy to use. Languages which are interpreter based are typically easy for novices

to use than compiler based languages. Interpreter based languages can be highly interactive, and some will examine each line of code as it is typed in, determine if the line is syntactically correct, and immediately inform the programmer if errors in syntax have been made. Compiled languages, on the other hand, can cause a beginning programmer a good deal of frustration, particularly when the compilation process frequently terminates due to detection of the inevitable syntactical errors made by novice programmers.

Gerhold (1980) insists that if teachers are to produce CAI, the "language must be interpretive", and that teachers should have to deal very little with such considerations as "alternative notations...memory allocation...variable type definition, and file management". (pp. 18-19)

In contrast, Bork (1980) suggests that the notion that any one language is much easier to learn than another is a myth, contending that "the way the language is taught is much more important than the language itself in determining the speed of initial learning." (p.41)

Structured Programming

Many computer scientists consider computer programming to be an art rather than an exact science. A problem is rarely capable of being solved only by a single, unique algorithm. Generally, the solution to a given problem may be arrived at by many different paths (algorithms). Furthermore, each different algorithm may be expressed as a variety of different computer programs, each with its own control structure, choice of variables and constants etc. Each of these resulting programs, although perhaps radically different in appearance, will ultimately produce the same result: a correct solution to the problem.

The inherent flexibility of this problem solving and programming approach has come under much criticism by many programming theorists, and much concern has been expressed over the lack of a rigorous, formal programming system. It has been suggested that an unstructured approach to the design of algorithms and computer programs can often result in the ultimate production of computer programs which,

although workable, are virtually unreadable and as such, tend to be both inefficient and difficult to modify.

This problem of lack of rigorous program structure has been specifically traced to the undisciplined use of the unconditional branch (the GOTO statement in BASIC). The careless use of such a programming feature can easily lead to the production of programs with a control structure which can only be described as 'spaghetti-like'! (Atherton, 1979)

Concern over the lack of a more formal programming system has led to the development of a number of computer languages which are highly structured, which discourage (or even forbid) the use of unconditional branching, and which facilitate programming using the technique of top-down problem solving.

Top-down problem solving involves taking a problem and dividing it into a number of very general sub-problems. Each sub-problem is then further divided into simpler sub-problems, and these sub-problems are further divided into still simpler units. This process of refinement ('divide and conquer'), moving from the general to the specific, eventually results in a problem being divided into a series of many easily-managed individual tasks. Grogono (1978) describes the top-down approach as follows.

> A program will require high level procedures to carry out large difficult tasks, such as 'solve a set of simultaneous equations', and these will call procedures at lower levels in the hierarchy, down to primitive procedures at the lowest level such as 'read a character'. The high level procedures will express the general and abstract features of the problem solution, and the low level procedures will contain the details of the implementation. The first important decision we must make when designing a new program is whether to start at the top of the hierarchy (the most abstract level), or to start at the bottom (most detailed level) and work up. Following the advice of a number of distinguished authors, we advocate the former method: top-down

> program development. Writing the program consists of proposing a very abstract solution and then successively refining it until it is entirely expressed in the chosen programming language. (p.278)

However, Grogono later cautions the reader that although top-down design is

> a method which is well documented, reliable and safe...it does not always yield efficient solutions, and it does not always provide deep insight into the nature of a problem. (p.280)

A good analogy for a top-down approach is provided by Zaks (1981) who suggests that an architect who begins a project by drawing rough sketches of the proposed building is using a top-down approach, wheras an architect who begins a project by consulting a brick catalogue is using a bottom up approach. In a similar way, a programmer who begins to design a chess playing program by typing:

```
10 INPUT "DO YOU WISH TO PLAY BLACK OR WHITE?";A$
20 IF A$ = "WHITE" THEN GOTO 100
```

is taking a very clear bottom up approach and will almost inevitably run into severe logic and control problems very quickly!

Listed on the following page is a slightly modified version of a "Moon Landing" program used by Atherton (1979) as an example of unstructured programming style. By examining the program closely, try to determine how it works, what the various variables represent, and what the calculations actually do. Note that arrows on the left of the program listing show the results of unconditional branching, and arrows on the right illustrate conditional jumps.

```
 20 PRINT"TIME","HEIGHT,"VEL","FUEL","BURN"
 30 GO=1.62:M=26000:D=10000:F=13000:T=1:T1=0:S=1
 33 P=1125:R=3:V=100:U=100
 40 PRINT T1,D,V,F,:INPUT B,D2:B=ABS(B)
 45 FOR X=S TO D2 STEP S:R2=1.7E+06
 47 G=GO-2*D/R2:IF D<1.0E+07 THEN 50:PRINT "TOO FAR":STOP
 50 V=U+T*G-(B*P*T)/(M+(M-T*B))/2):T1=T1+T
 60 M=M-(T*B):D=D-((U+V)/2*T):U=V:F=F-B*T
 65 K=K+(B*.23)-9.87854:IF P*B+G*100>2.5E+5THEN220
 70 IFD<=0 THEN110:NEXTX:IFK>2000 THEN200:IF K>1500 THEN210
 75 IF F>0 THEN 40
 90 PRINT "OUT OF FUEL AT";T1:B=0:S=1.0E-3:GOTO45
110 PRINT "ON MOON AT";T1;"SECONDS. LANDING VELOCITY";V
120 IF V<10 THEN 160:IF V<20 THEN 170
130 PRINT "ALL DEAD,BLASTED CRATER";V*11.78;"KM.":STOP
160 PRINT "SAFE LANDING":STOP
170 PRINT"CREW INJURY";INT(3.2*V/17.46);"BONES BROKEN":STOP
200 PRINT "TUBE BURNOUT";R-1;"LEFT":R=R-1:IF R=0 THEN 207
205 P=P/(4-R):GOTO 75
207 PRINT "ALL TUBES GONE":S=1.0E-3:B=0:GOTO 45
210 PRINT "POWER TUBES TOO HOT":GOTO 75
220 Z=RND(0)*30:PRINT "BLACK OUT FOR";INT(Z*D2);"SECS."
230 D2=Z:B=25:GOTO 45
```

The undisciplined use of both types of branching, the use of multiple statements per line, and the lack of any programmer comments results in a program that is extremely difficult to follow.

If the program shown above is rewritten in Pascal, using top-down programming techniques, the resulting program will be much easier to understand. Furthermore, the program will be easy to modify at a later date. Partial listings for a Pascal "Moon Landing" program are provided on the following page. Please note that the procedures presented here are designed for maximum clarity, as opposed to maximum efficiency.

The main program itself consists only of a few lines...

```
BEGIN (*main*)
  REPEAT
      initialize;
      printinstructions;
      REPEAT
        calculate;
        IF NOT safe THEN safetycheck;
      UNTIL madecontact;
      playagain;
  UNTIL bored
  writeln ('Bye for now!!')
END. (*main*)
```

There are two main blocks in this program. The outer block repeats until the user is bored of playing the game, and the inner block repeats until contact is made with the surface of the Moon.

The first two procedures initialize variables and print user instructions, and then calculation begins.

The calculate procedure (listed on the following page) determines new variable values, and determines if conditions are safe.

The safetycheck procedure (listed below) is invoked when unsafe conditions arise.

```
PROCEDURE safetycheck;
BEGIN (*check to see what the problem is*)
  IF height > maxdistance THEN lostinspace;
  IF       (fuel<=minfuel)
        OR (tubes = minpower)
        OR (height <= mindistance)
  THEN mustland;
  IF temperature > deadlyheat then altertemperature;
  IF acceleration > maxpain then blackout;
END; (*end of checking procedure*)
```

```
PROCEDURE calculate;
BEGIN (*calculate procedure*)
  REPEAT
      newgrav:=gravity-2*height/rvalue;
      speed:=newspeed+newgrav-(burn*power)/(m+(m-burn))/2;
      time:=succ(time);
      m:=m-burn;
      height:=height-(newspeed+speed)/2;
      newspeed:=speed;
      fuel:=fuel-trunc(burn);
      temperature:=temperature+(burn*burnvalue)-coolvalue;
      acceleration:=power*burn*newgravity*gvalue;
      count:=count+increment;

      IF       (height>maxdistance)
            OR (fuel<minfuel)
            OR (temperature>deadlyheat)
            OR (acceleration>maxpain)
            OR (height<=mindistance)
      THEN safe:=false;
  UNTIL (count>rate) OR (NOT safe);

      IF (temperature>maxheat) AND (temperature<deadlyheat)
      THEN writeln('Power tubes are overheating!!');

END; (*calculate procedure*)
```

All of the procedures that can be invoked within the safetycheck procedure (ie. lostinspace, mustland, altertemperature and blackout) carry out the specific task which their name suggests. For example, the altertemperature procedure will provide the user with an appropriate diagnostic message, and then adjust both the power output and temperature of the powertubes. This procedure is listed overleaf.

```
PROCEDURE altertemperature;
BEGIN (*temperature adjustment*)
  writeln ('Heat excessive...power tube burned out!');
  tubes:=pred(tubes);
  IF tubes = minpower THEN
      BEGIN (*all power is lost*)
        writeln ('You now have no power tubes!');
        powerleft:=false;
      END (*all power is lost*)
  ELSE
      BEGIN (*power is reduced*)
        writeln ('you now have ',tubes,' tubes left.');
        power:=power/(4-tubes);
      END; (*power is reduced*)
  temperature:=normaltemp;
END; (*temperature adjustment*)
```

Finally, the playagain procedure simply determines if the user wishes to play the game again.

```
PROCEDURE playagain;

VAR
   answer:char;
   yes:boolean;

BEGIN (*want to play again?*)
  write ('Another game of Moon-Lander ?');
  readln (answer);
  yes:=answer IN ['Y','y'] ;
  IF NOT yes THEN bored:=true;
END; (*want to play again?*)
```

It should be evident that a language which supports the techniques of structured programming will enable a programmer to produce much more readable code. The Pascal Moonlander program was written without any unconditional branches, and all procedures have single entrance and single exit points. Furthermore, via the use of meaningful variable, procedure, and constant names, the portions of the program listed above should be comprehensible even by

those readers who do not know Pascal.

Some readers may complain that the Pascal program is so much longer than the BASIC program. However, unless available memory space is at a real premium, readability should never be sacrificed in order to gain a few bytes of memory.

Languages which support the development of programs via a top-down problem solving approach are particularly useful when a team of programmers will be utilized. In this way, each member of the team can be assigned an easily managed sub-problem and, with the use of local variables, can then create a free-standing, fully tested program block. At a later date, the completed network of program blocks which has been generated by the team can be assembled into a logically flowing main program.

## Language Descriptions

High level computer languages can generally be categorized as belonging to one of four main classes:

(a) procedure-oriented languages
(b) problem-oriented languages
(c) list-processing languages
(d) actor languages

Procedure oriented languages are often considered to be languages of a rather general purpose nature. Programs are written as a series of routines (functions, procedures, blocks, subroutines etc.), often in a form which closely resembles English. When using procedure oriented languages, programmers need not concern themselves with the inner workings of the computer (unlike low level languages), but need to focus only on the design of the procedures themselves. Programs written in ANSI-standard versions of the procedure oriented languages tend to be relatively portable. Some languages which fall into this category are highly structured and block oriented (Pascal), whereas others are much less structured (BASIC).

Problem oriented languages, in contrast to the procedure oriented languages, are designed to perform a highly specific programming purpose. These purposes extend from music creation (MAX) to courseware creation (PILOT). Since these languages tend to be so task-specific, programmers are free to concentrate on developing algorithms, leaving the computer to develop many of the program procedures. In this way, the efficiency of the problem solving process is enhanced. Problem solving languages tend to be rather inflexible, are rarely very portable, and are not usually configured to run on microcomputers.

List processing languages, as the name implies, are designed chiefly to be facile manipulators of strings of non-numeric information. As such, they are mainly used to tackle the difficult problems which arise in the artificial intelligence community (robotics, pattern recognition, natural language processing, general problem solving, etc.). Although most list processing languages are too

massive to run on microcomputers, some versions (LISP) have recently been released.

The Actor languages represent a radical departure from the concepts of the more traditional programming language types described above. In the Actor languages, the programmer does not write serially-processed, procedure-oriented programs, but instead creates parallel-processed, object-oriented programs. In these languages, the programmer writes (directs) a program (screenplay) in which objects (actors) exhibit behaviours according to pre-defined instructions (scripts) only when they receive the appropriate messages. Furthermore, actors only take up memory space when needed, and if the action required by a particular message is not a part of their behavioural repertoire, the message can be passed on to another nearby actor, who may also,if necessary, pass the message on to another actor. Program design in the Actor languages has been described as being "totally upside-down and inside-out from all other computer languages". (Nelson, p.62) The Actor languages are well suited to solving problems in both the artificial intelligence and CAI communities, may be either interpreted or compiled, often use virtual memory techniques, have spectacular graphics capabilities, and tend to be very fast. They may just revolutionize computing.

## FORTRAN

This procedure oriented language was, for many, a first introduction to the world of computers. Created by John Backus, it was introduced in 1957 and, as its name (FORmula TRANslator) suggests, it was designed with scientific and mathematical applications in mind.

Originally developed to run on the IBM 704 machines, FORTRAN has since evolved into many different versions. Two of the most popular of these versions were developed in Canada at the University of Waterloo (WATFOR and WATFIV). While most mainframe versions of FORTRAN run in batch mode, some interpreter-based interactive dialects exist, and microcomputer versions are becoming increasingly available.

The newer versions of FORTRAN (the "standard" version is FORTRAN 77) support structured programming techniques, and programmers can link prewritten subroutines to their own programs with relative ease. One of the biggest advantages of FORTRAN is that a huge bank of pre-written subroutines has been established over the years, thus making life simpler for the programmer.

FORTRAN, however, does not have particularly strong text processing capabilities and therefore remains a language best suited to program development in the areas of math, science and engineering. It is not well suited to the development of CAI programs.

## COBOL

COBOL (COmmon Business Oriented Language) was introduced in 1960 and shortly thereafter adopted as the "official" language of the U.S. Department of Defence. As such, COBOL has become the most commonly used of all computer languages. However, although microcomputer versions are now available, the language remains a language primarily suited to the business and data processing community, and holds little interest for those who wish to develop CAI programs.

## LISP

Introduced in 1961, LISP (LISt Processor) was developed by John McCarthy of the Massachussets Institute of Technology. LISP is of primary interest to those in the artificial intelligence community, and is capable of handling large quantities of data (primarily non-numeric data) and of performing pattern matching and string-substitution analyses on the data. A microcomputer version is now available and the language may be of use to those interested in the development of sophisticated dialogue-type CAI packages.

## BASIC

Developed by John Kemeny and Thomas Kurtz of Dartmouth College, and introduced in 1965, BASIC was envisioned as a quick, easy to learn introduction to the world of computer programming.

BASIC (Beginner's All-purpose Symbolic Instruction Code) is a highly interactive, interpreter-based language, and a computer novice can expect to learn how to write programs within a matter of a day or so. This ease of learning has prompted one author (Schuyler, 1979) to state that

> BASIC is an excellent language to use when the task at hand is to teach programming. And it is also excellent for small to medium-size programs of limited complexity. (p.29)

Perhaps because it is so easy to learn, BASIC has become a popular favourite amongst hobbyists and educators. BASIC is also, from a technical point of view, an easy language to implement on a small machine, and some minimal versions of the language can run on as little as two kilobytes of memory. This ease of implementation has resulted in versions of BASIC being available for virtually every brand of microcomputer on the market, and this

accessibility to the general public has further increased the popularity of the language.

BASIC, however, has serious drawbacks, perhaps the most serious of which is due to the very popularity of the language itself. Atherton (1979), in speaking of progress in the development of CAI, states that

> The major school language is BASIC and here lies one of the major obstacles to progress....It has become many languages as manufacturers have competed to extend the original concept into more sophisticated dialects. The worst of these positively demand bad programming styles and even the best have serious defects. (p.205)

As suggested by Atherton, BASIC is not a language well suited to promotion of the concepts of structured programming and, unless the programmer is highly disciplined, large programs can easily become unwieldy and virtually impenetrable.

However, if portability is not a concern, the language may still be a reasonable choice for a courseware development team, particularly if the CAI programs will limited in both size and complexity. In addition, there are some newer versions of the language now available which do have the control structure necessary for structured programming. Furthermore, for those who place a high premium on execution speed and memory utilization, some of the newer dialects are also compiler based.

**APL**

Created at Harvard University by Canadian, Ken Iverson, APL was not originally intended to be a computer language. It was, instead, intended to be a new system of mathematical notation, a system devoid of the logical inconsistencies that are present in our traditional system of mathematical notation.

Iverson eventually came to IBM, however, and APL (A Programming Language) was born, and eventually introduced

to the public in 1967.

To many people, APL, is most familiar as the language with the bizarre looking character set, and odd way of constructing program statements (right to left, rather than the more traditional left to right approach!). The language, however, is extremely powerful, having remarkable array processing capabilities, and the ability to handle recursive functions with ease. It is also an exceptionally dense language, with single operators carrying out functions that would require many lines of code in other languages.

APL is sometimes criticized for its lack of a formal, rigidly defined control structure, but unconditional branching is rarely used in APL programs, and the modular design of APL programs encourages good programming technique. Other criticisms stem from the fact that APL is almost always interpreted and thus, to some, likely to be somewhat slow. However, the very power of the language allows very high execution speed even though the code is not compiled, and the question of interpretive overhead is therefore not particularly relevant. There are those who suggest that APL is a difficult language to learn, but considerable evidence, albeit subjective and somewhat anecdotal, suggests that APL is only difficult to learn if the programmer has already acquired bad programming habits as a result of previous exposure to languages like BASIC. Probably the most valid criticism of APL is that there is an unfortunate tendency for APL programmers to produce code that is so compact and efficient that it is virtually unreadable by anyone other than the author!

Until recently, APL was primarily a large-system language, but subsets of the language can now run on several brands of microcomputer. The only complete version of the language to run on a microcomputer to date, however, is the one (designed by Ken Iverson and Ted Edwards) which runs on the APL machine manufactured by MCM.

Pascal

Pascal is the language envisioned by many as the

eventual replacement of BASIC as the most popular microcomputer language. Named after the French mathematician Blaise Pascal, it was designed in Zurich by Nicklaus Wirth in 1968, and first introduced in 1971. A revised version, henceforth known as Standard Pascal, was eventually released in 1973.

Pascal has many strong points, probably the best known of which is that it fully utilizes the concepts of structured programming. Programmers used to less structured languages often experience some difficulty when switching to Pascal. Unlike BASIC programs, it is not possible to casually add variables to Pascal programs as an "afterthought". This is because, in Pascal, all variables must be defined and declared prior to use. In fact, since all data types (variables, constants, files, procedures, functions etc.) must be declared and defined prior to use, Pascal virtually forces the programmer to adopt a disciplined programming style.

The fundamental building blocks of Pascal are procedures and functions. Procedures loosely correspond to subroutines in BASIC, except that, unlike subroutines, procedures can use their own (local) data types which are distinct from the (global) data types used by the main program. This enables the programmer to write and test procedures as separate entities. This, coupled with the ease with which functions and procedures are invoked in Pascal, further encourages the programmer to adopt the principles of top down programming.

Pascal allows the use of data type names of any length, although only the first eight characters are ordinarily recognized by the system. Furthermore, programmer comments may be placed anywhere within a program. These characteristics of Pascal programs, when combined with clearly defined data types and a nicely spaced and formatted program outline, tend to make Pascal programs extraordinarily readable. In fact, Pascal is often proclaimed to be a "self-documenting" language.

Pascal is also a highly portable language due to the use of what is known as p-code. In most other compiler-based languages, source programs are directly compiled into machine executable code, and the compiler

must therefore be written in the native code of the host computer. When a Pascal program is compiled, however, it is first compiled into an intermediate code known as p-code, and the host computer then interprets the p-code. Although the extra step of interpreting the p-code means that Pascal programs do not execute as quickly as programs written in the more traditional compiler-based languages, the added benefit of such a step is that only the p-code interpreter need be written in the machine language of the host computer, and this, when compared to the task of writing a full compiler in the host language, is relatively easy. This all means that a Pascal program, once it has been compiled to p-code, can be run directly on any other computer that has its own p-code interpreter, thus making the language very portable.

Standard Pascal is sometimes criticized as being inappropriate for use as a CAI language due to its lack of intrinsic string handling functions. However, UCSD (University of California at San Diego) Pascal does support such functions and is therefore a much better choice for CAI development.

In summary, Pascal's portability, ease of learning, support of structured programming techniques and high execution speed all combine to make Pascal (particularly UCSD Pascal) an attractive choice for CAI development.

## PILOT

PILOT, first specified by John Starkweather and Marty Kamp in the early seventies, is a problem-oriented language specifically designed to facilitate the development of CAI programs.

Since PILOT is interpreter based and highly interactive, it is very easy to learn. In addition, PILOT handles natural language learner responses with ease (including such facilities as the automatic detection of simple spelling errors), and is independent of any particular teaching strategy.

Unfortunately, since PILOT is interpreted through a

high level host language (ie. BASIC for Hewlett Packard PILOT, Pascal for Apple PILOT), it tends to have a slow execution speed. This, combined with its inability to rapidly handle computations of any complexity, makes it a poor choice for the design of simulation CAI, since this type of CAI often demands high execution speed and the ability to easily perform complex mathematical computations.

However, with its powerful input handling capabilities, PILOT is an excellent choice for drill and practice, tutorial and dialogue CAI.

SMALLTALK

Many of the high level languages described above can be described as being composed of two fundamental structural entities:

| | |
|---|---|
| Data Structures: | the relationship between sets of data items. |
| Procedures: | A set of unambiguous rules for the solution of a problem in a finite number of steps. |

Languages based on this dual entity concept are inherently sequential in nature, dealing with one task at a time. Furthermore, when large and complex programming tasks are undertaken, these systems of procedures and data structures have the unfortunate tendency to become cumbersome, unwieldy and unreliable.

In addition, the way in which a high level language is structured is profoundly important in that the structure strongly influences the way in which one approaches the development of a programmatic solution to a program. Research suggests that human thought processes are not sequential, but are instead parallel in nature. It seems reasonable to suggest that a language whose structure supports the parallel processing of tasks could therefore facilitate the translation of human problem solving techniques into computer problem solving techniques.

With these ideas in mind, Alan Kay et al. of the Learning Research Group at Xeroc PARC (Palo Alto Research Centre) set about developing a language whose inherent structure would be vastly different from the traditional languages described above. Guided by the concepts of the SIMULA language, the extremely powerful and radically new language known as SMALLTALK was created. Concomitant with the development of the language was the development of the DYNABOOK idea. The notion is that SMALLTALK will eventually run on a textbook sized personal computer, complete with a flat gas plasma display screen. The current technological advances being made towards electronic minituarization suggest that this idea is certainly one that can be realized in the near future.

The structure of the SMALLTALK language replaces the highly specific dual entity concept of the traditional languages with a much more general structure known as an activity. These activities only exhibit behaviours when "stimulated" or acted upon by appropriate messages. Activities can both send and receive messages. A message-activity language such as SMALLTALK is therefore parallel in nature, with every activity prepared at all times to send messages to, and receive messages from other activities. The entire language can be thought of as a series of activities dynamically interacting in parallel. Just as the traditional languages are thought of as being primarily hierarchical, SMALLTALK can be thought of as a heterarchical language

Similar activities are said to belong to the same family, and a family is described by a "dictionary" of traits, as in the example below.

| FAMILY | TRAITS |
|---|---|
| circle | grow |
| | turn |
| | flip |
| | flatten |
| | stretch |
| | pop |

The programmer is able to modify the dictionary of traits for any given family. For example, the trait "shrink" could be added to the dictionary of traits illustrated above. Furthermore, new families can be created by modifying and combining traits already existing for other families.

The grammar of SMALLTALK is relatively simple. The language is made up of phrases, with each phrase being a message to an activity. A description of the activity is followed by a message which selects the particular trait the programmer wishes to see performed. There are strict limitations governing which messages are appropriate to which activities, and the activity must therefore "decide" whether or not to accept the message. Take the following as an example.

SOCRATES SHRINK 18

This phrase tells the computer first of all to locate the activity named Socrates. As it turns out, the computer would discover that the activity named Socrates is a member of the family "circle". The computer would then scan down the list of traits associated with the circle family until it located the trait "shrink". The circle named Socrates would then obediently shrink 18 screen units.

SMALLTALK is capable of using a display screen in a fascinating manner. Since the screen is often not large enough to simultaneously display all of the information a user wishes to scrutinize, the display is broken down into overlapping "windows" of information. These windows, similar in appearance to a stack of sheets of paper, can all be made to be at least partially visible at all times. A simple command can cause any chosen window to move to the top of the stack, overlapping the others. Windows which the user feels will not be utilized for some time can be shrunken down to a convenient size, labelled for future reference, and neatly tucked in a handy corner of the display screen. Another simple command will cause the window to instantly expand and move to the top of the stack, once again revealing its contents.

SMALLTALK, for a long time shrouded in mystery, may soon be available to the public. A microcomputer version

of SMALLTALK has been developed by a company called Rosetta, and it may be on the market soon. The Rosetta version of SMALLTALK is designed to run on the Exidy Sorcerer microcomputer under CP/M.

Readers wishing to learn more about this language are referred to the September 1977 issue of Scientific American, the October 1980 issue of Creative Computing, and the September 1981 issue of Byte magazine.

# APPENDIX II: GRAPHICS ON THE APPLE II

# GRAPHICS AND ANIMATION WITH THE APPLE II

## Introduction

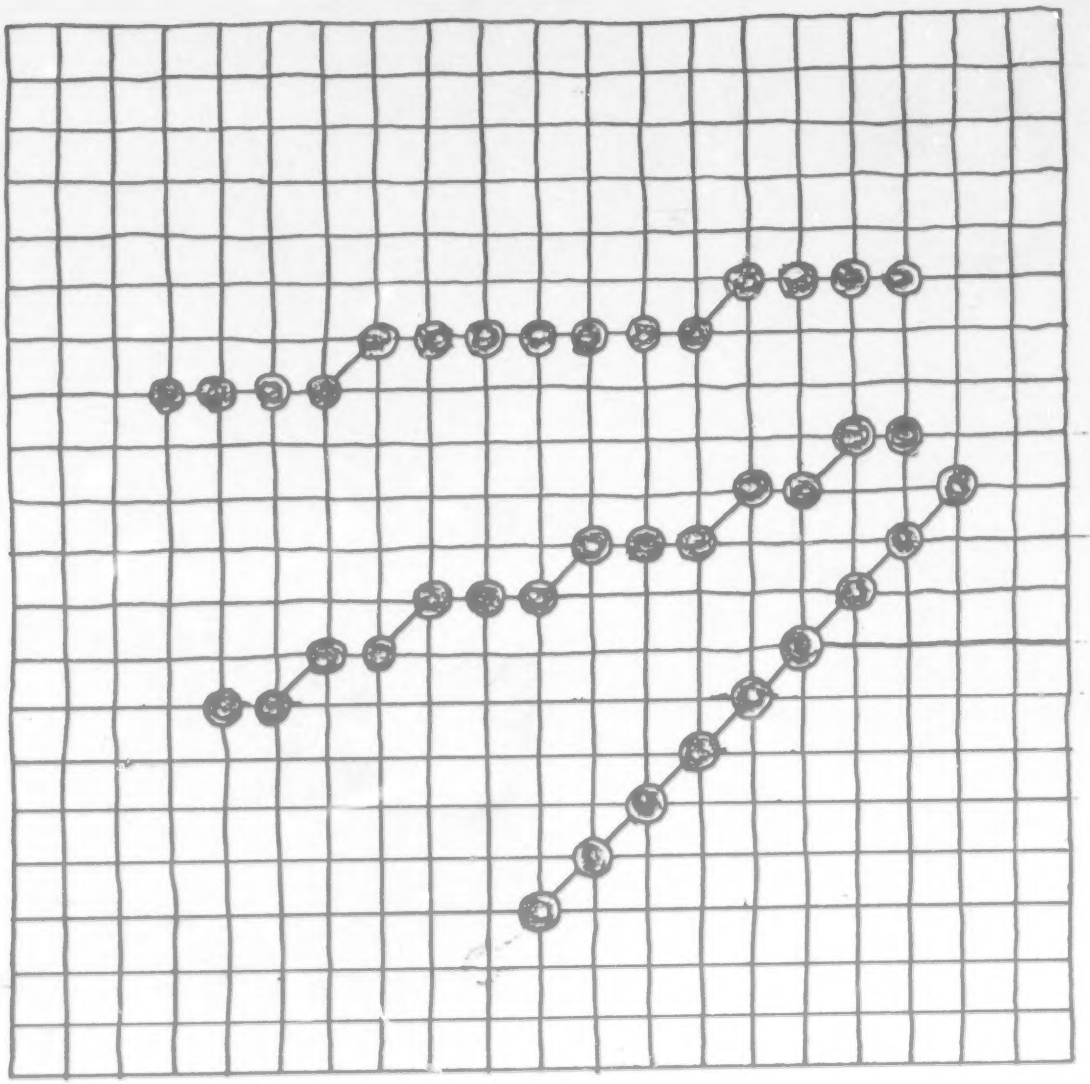
The type of display device used with most microcomputers is known as a raster device. Television sets and monitors are both raster devices, with the display area being made up of a matrix of pixels (plottable points). The number of pixels which are available for illumination determines the resolution of the display device. One can imagine a matrix of pixels as being analogous to a sheet of graph paper upon which can only be plotted at the intersection points of the grid lines. The figure on the following page illustrates why it is not always possible to plot perfectly straight lines on a raster device. The stepwise appearance of some of the "straight" lines is due to that pixels do not always "exist" at the points the programmer wishes to illuminate. A higher resolution display with more pixels (i.e. finer graph paper) can reduce this stepwise appearance. Some hardware supports the use of grey scale algorithms which can fine-tune the appearance of these lines by appropriately adjusting the brightness of certain of the pixels. The improved appearance, however, is gained only at the expense of a decreased execution speed.

A more sophisticated display can be generated on a vector device. These display devices are much more expensive, but are not pixel based and therefore allow the programmer to illuminate points at any location on the display screen. In this fashion, perfectly straight lines can be drawn between any two points.

The Apple supports two graphics modes. They differ in resolution, colour, and the amount of RAM required to illuminate the screen.

In low resolution mode, the programmer can control the illumination of 1600 individual rectangles on a 40x40 grid. Each rectangle is made up of 12 individual pixels arranged in a 3x4 matrix.

Low resolution mode is entered by issuing the GR



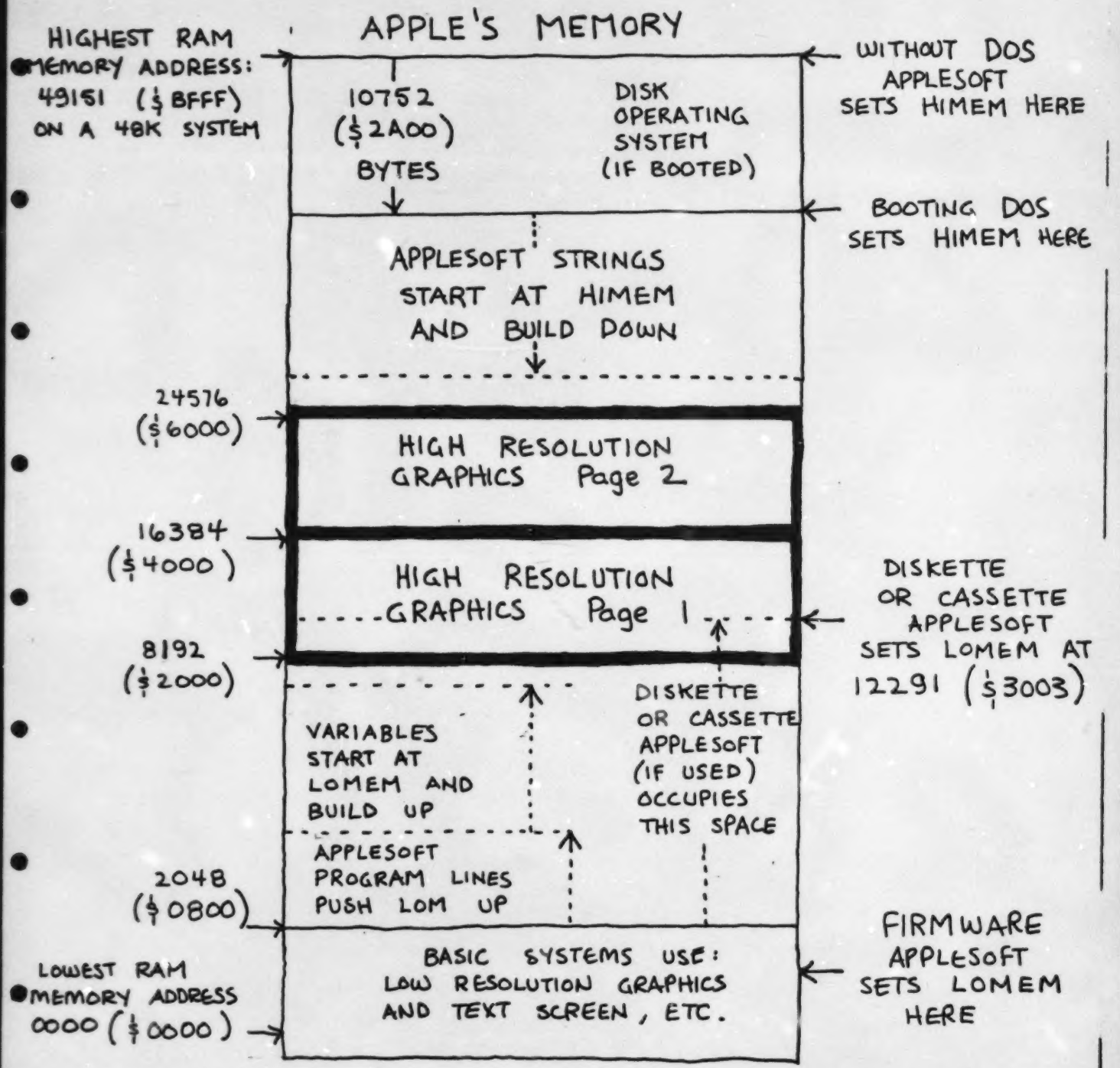
instruction and specifying a colour via the COLOR instruction. For example, the instruction COLOR=9 will set the colour to orange. It is possible to plot all 16 different colours on one screen. A single rectangle may be illuminated by using the instruction PLOT X,Y with X and Y representing coordinates on the 40x40 low resolution grid. Horizontal lines may be displayed by using the HLIN instruction. For example, HLIN 4,17 AT 14 will draw a horizontal line at row 14 from columns 4 to 17. Vertical lines may be displayed in a similar manner using the VLIN instruction. Low resolution graphics uses the same 1K of RAM as the text screen.

There are two high resolution pages in Applesoft, each of which occupies 8 kilobytes of memory. The programmer specifies which page is to be used by issuing either the HGR or the HGR2 instructions. When using page 1, the programmer is automatically provided with four lines of text (from text page 1) at the bottom of the screen. If the entire screen is to be used for graphics, the instruction POKE -16302,0 will remove the four text lines and replace them with the appropriate area for graphics.

Page 2 of graphics automatically provides the programmer with a full screen for graphics. If the programmer wishes to reserve four lines at the bottom of the screen for text, then text page 2 must be utilized. Using text page 2 is not clearly explained by the Apple company, and readers are referred to the July 1981 issue of Microcomputing magazine for an informative article on this subject.

Table 5 (soft screen switches) of the Apple II Reference manual (p. 130) describes the eight POKE instructions that can be used to set the possible combinations of graphics modes.

The Apple memory map illustrated on the following page shows where the high resolution pages reside in memory. Programmers sometimes discover that, when using page 1, a large program may cause a visual disruption of the graphics display. This is because a large program can eventually "bump" into the bottom of the high resolution page. The easiest solution is to use page 2 of graphics instead. A glance at the memory map shows that, by using page 2, the


APPLE'S MEMORY
HIGHEST RAM MEMORY ADDRESS: 49151 ($BFFF) ON A 48K SYSTEM
10752 ($2A00) BYTES
DISK OPERATING SYSTEM (IF BOOTED)
WITHOUT DOS APPLESOFT SETS HIMEM HERE
BOOTING DOS SETS HIMEM HERE
APPLESOFT STRINGS START AT HIMEM AND BUILD DOWN
24576 ($6000)
HIGH RESOLUTION GRAPHICS Page 2
16384 ($4000)
HIGH RESOLUTION GRAPHICS Page 1
DISKETTE OR CASSETTE APPLESOFT SETS LOMEM AT 12291 ($3003)
8192 ($2000)
VARIABLES START AT LOMEM AND BUILD UP
DISKETTE OR CASSETTE APPLESOFT (IF USED) OCCUPIES THIS SPACE
APPLESOFT PROGRAM LINES PUSH LOM UP
2048 ($0800)
FIRMWARE APPLESOFT SETS LOMEM HERE
LOWEST RAM MEMORY ADDRESS 0000 ($0000)
BASIC SYSTEMS USE: LOW RESOLUTION GRAPHICS AND TEXT SCREEN, ETC.

programmer is given an extra 8 kilobytes of RAM for program storage. In fact, just as a matter of convenience, many programmers automatically decide to use page 2 of graphics, regardless of how large the program will be.

The question of resolution when using High resolution graphics often causes considerable controversy.

There are essentially 560 available dot positions in each horizontal line. It can be said, therefore, that black and white high resolution graphics mode provides a 560x192 resolution capability. In colour high resolution mode the resolution works out to be, in effect, 140x192 in six colours. Working from the leftmost dot position, the 560 available dots are cyclically illuminated in the following manner; violet, blue, green, orange, violet, blue, green, orange, etc. It can be seen that, in this way, only 140 colour points exist, each one being made up of 4 consecutive dots. White is obtained when any two consecutive even or odd dots are simultaneously turned on. This is why 2 whites exist; White 1 appears when even consecutive dots are turned on, and White 2 appears when odd consecutive dots are turned on. In a similar manner, Black 1 and Black 2 occur when these dots are not turned on.

The following program (adapted from the Fall 1980 issue of The Apple Orchard) will first draw a diagonal line in "280 point" mode,then draw a series of lines (parallel to the first) in "560 point" mode. The spacing of the parallel series of lines can be modified by the user, and manipulation of this parameter will produce unusual effects.

```
1     TEXT:HOME:INPUT "SPACING";TEST
10    HGR
30    HCOLOR=3:HPLOT130,0 TO 140,159
90    FOR Q=280 TO 300 STEP TEST
100   FOR Y=0 TO 159
110   X%=Q+Y/8
120   Y%=Y
130   GOSUB 1000
140   NEXT Y
145   NEXT Q
150   GOTO1
1000  XX%=X%/4: M%=X%-4-XX%
1040  IF M%=0 THEN HCOLOR=2
1050  IF M%=1 THEN HCOLOR=6
1060  IF M%=2 THEN HCOLOR=1
1070  IF M%=3 THEN HCOLOR=5
1080  HPLOT X%/2,Y%:RETURN
```

Creating Shape Tables

A high resolution shape can easily be displayed upon the screen by the utilization of a series of HPLOTs. However, if the particular shape is one that is to be used frequently, if it is one that must be displayed rapidly, if it is one that must be selectively erased, or if it is one that must be moved around the screen, then the creation of a shape table for that shape is the most expedient way of accomplishing the task.

A shape table may contain up to 255 high resolution shapes with each one of these shapes subsequently being referred to by a number indicating its position within the table. Once a shape has been defined and stored in a table, it can be displayed by using the DRAW statement. For example, DRAW 5 AT 30,20 will cause the computer to display the fifth shape in the table at the X,Y coordinates 30,20. In a similar manner, XDRAW 5 AT 30,20 will selectively erase the shape. The display size of a predefined shape can be altered by using the SCALE

statement, the scale may range from 1 to 255. Furthermore, the shape can be rotated through 64 possible angles on the screen, by using the ROT statement. For example, SCALE=12 sets the scale at twelve and ROT=Ø sets the rotation at zero. It should be noted that when SCALE=1, only four rotation values are recognized (Ø,16,32,48 representing Ø,9Ø,18Ø,27Ø degrees). When SCALE=2 then eight rotation values are recognized, etc. Furthermore, some distortion of the shapes is to be expected when certain rotation values are chosen.

There are four distinct ways of defining shape tables; three of which require special software support.

1. Pencil and Paper

Shapes are made up of straight line segments, with the straight lines defined by unit vectors. These vectors (up, down, left, right) are placed end to end until the complete shape is defined. A complete, detailed description of this method may be found in the Applesoft II Basic Programming Reference Manual (p. 92) and in the Computer Graphics Primer (Waite, p.129).

2. Keyboard

A much simpler method of shape table definition involves using keyboard strokes to create the shape. The MECC programmer's aid diskette contains a program called TABLES which accomplishes this. The user simply runs the program and specifies a colour and a starting point for the shape. The starting point is defined by a pair of X,Y coordinates, with the screen being divided into a 279x159 coordinate grid. From this point, simple keystrokes move a point of light around, leaving an illuminated trail which outlines the shape. The keys U,D,L,R,B,P,N stand for up, down, left, right, backup, plot and noplot respectively. Detailed instructions for using the program accompany the diskette.

3. Paddles or Joystick

A program exists which allows the user to draw a shape on the video screen using two paddles or one joystick. The shape is drawn freehand in "etch a sketch" fashion, or, if freehand drawing is not accurate enough for the given application, a keyboard mode allows you to use the U,D,L,R,B,P,N keys to define the shape. This program is available from Rocky Mountain Software Inc., #101-2044 West Third Ave., Vancouver, B.C. V6J 1L5 (Tel:604-266-7684). The program is available for $45.00.

4. Graphics Device

A program exists which allows the user to draw a shape on the video screen by using the Apple graphics tablet. Once agin, if freehand drawing is not sufficiently accurate, a keyboard mode also exists. Several other features may be found on this diskette which allows the user to place shapes within a graphics tablet picture, do simple animation, etc. This program is also avaiable from Rocky Mountain Software Inc., at a price of $99.00.

Furthermore, the Versawriter graphics generating device is capable of generating shape tables in a similar manner, and the necessary software is supplied with the drawing tablet.

When using shape tables in a program, it is necessary to load them into memory (with the BLOAD instruction) at an appropriate place. Many programmers prefer to load shape tables at a memory location just above high resolution page two. It is also necessary to indicate to the system the address of the shape table before the table can be accessed. This is accomplished via the use of two successive POKE instructions, as follows:

POKE 232, low order of table address and

POKE 233, high order of table address

The values for low and high order are determined by dividing the table address by 256. In this way the resulting dividend is the high order and the remainder is the low order. For example, if you wished to load the shape tables at memory location 24576, then the two pokes would be:

POKE 232,0

POKE 233,96

## Animation

Animation on the Apple II is conceptually similar to the old fashioned "flip books" in which one simply flips through various pages, each page illustrating an updated image. In this manner a flickering illusion of movement is obtained, with the degree of flicker being determined by the rate at which one flips the pages. On the Apple, this "flipping" is done electronically, with the amount of flickering being determined, at least partially, by program execution speed. This explains why smoother animation is obtained when using Integer BASIC rather than Applesoft and why the smoothest animation is obtained when the program is written in machine code.

On the Apple, animation occurs when an image is displayed, erased, updated, displayed, etc. There are three ways in which the image can be displayed. It can be displayed as a series of HPLOT's, as a shape in a shape table, or as a series of user defined graphics characters. Each method will be dealt with separately:

### (i) HPLOT's

The program listed on the following page illustrates how one can cause a triangle to move across the display screen. The first GOSUB loads the X,Y coordinate data into two arrays. The variable VELOCITY controls the rate at which the horizontal position of the triangle is updated by the variable UPD. Line 30 draws the figure, line 40 erases the figure and line 35 causes the figure to linger on the screen in an attempt to reduce flickering.

The most serious problem with this technique is that too much time is taken to draw the figure, particularly if the figure involves a large number of coordinates. This makes for an annoying flicker in the display. We obviously must find a better method of displaying our figure!

```
10    GOSUB 900:VELOCITY=0:HGR
15    VELOCITY=VELOCITY+5
23    FOR UPD=200 TO 20 STEP -VELOCITY
30    HCOLOR=3:GOSUB 200
35    FOR PAUSE=1 TO 50:NEXT PAUSE
40    HCOLOR=0:GOSUB 200
50    NEXT UPD
60    GOTO 15
200   HPLOT UPD + X(1), Y(1)
210   FOR Q=2 TO 4:HPLOT TO UPD+X(Q),Y(Q):NEXT Q
220   RETURN
900   FOR X=1 TO 4:READ X(X):NEXT X
910   FOR Y=1 TO 4:READ Y(Y):NEXT Y
920   RETURN
1000  DATA 50,60,40,50,50,80,80,50
```

(ii) Shape Tables

Any given figure can be displayed much more quickly if it is first defined as a shape in a shape table. Once this has been done there are essentially three methods of animating the figure. The first one involves an alternating series of DRAWS and XDRAWS, as in the following program segment.

```
40 FOR INC = 10 TO 80 STEP 2
50 DRAW 12 AT 10+INC,40
60 XDRAW 12 AT 10+INC,40
70 NEXT INC
```

This works quite well unless there happens to be any overlapping line segments in the original defined shape. If this is the case, annoying "trails" will occur. This problem can be rectified by using another, albeit slower, animation technique as follows.

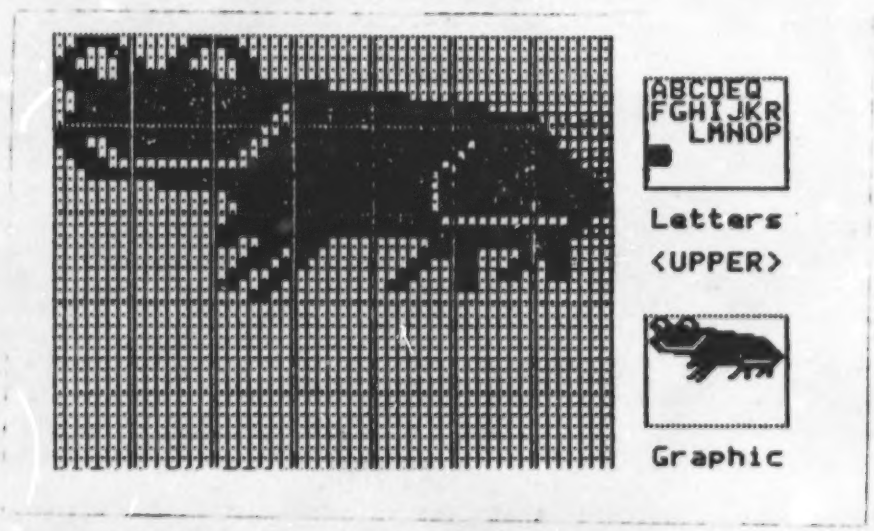
```
40 FOR INC = 10 TO 80 STEP 2
50 HCOLOR = 3:DRAW 12 AT 10+INC,40
60 HCOLOR = 0:DRAW 12 AT 10+INC,40
70 NEXT INC
```

Unfortunately, if there is a background against which your figure is moving (such as a graphics tablet picture), the area over which your figure moves will be erased! Both of the problems mentioned above can be solved with the third technique. A series of XDRAW's will not produce any trails and the integrity of the background will remain intact.

```
40 FOR INC = 10 TO 80 STEP 2
50 XDRAW 12 AT 10+INC,40
60 XDRAW 12 AT 10+INC,40
70 NEXT INC
```

(iii) Character Sets

It is possible to achieve rather smoothly flowing animation by means of the generation of programmable graphics character sets. Although there are various approaches to the problem, perhaps the easiest is simply to use the Apple II DOS Toolkit. This diskette holds a program which enables the programmers to generate character sets of their own choice using the game paddles. Those who are familiar with the children's game RIB*BIT may be interested to know that the pleasant animation in this program is a result of the generation of a graphics character set. Part of this character set is illustrated below.



Once the character set has been generated and saved, the user can write a program which will automatically load in the high resolution character generator and the programmer-defined character set. Once in memory, the programmer can define animation figures as elements in an array. For example, the frog previously illustrated is defined as follows:

```
F$(4)="ABCDEQ
       FGHIJKR
         LMNOP"
```

From this point on, printing F$(4) will cause the frog to appear on the high resolution screen, and altering the position at which F$(4) is printed (via VTAB and HTAB) will cause the image of the frog to move across the screen.

The Toolkit enables the programmer to easily alternate displays of graphics characters between both high resolution screens. In this way, when an image is displayed on the first screen, an updated image is being generated on the second screen. The screen display soft switch is then turned, and the second screen is displayed while an updated image on the first screen is generated. This facility (sometimes referred to as "ping-ponging") reduces flicker to a minimum, resulting in exceptionally smooth animation.

(iv) Three-Dimensional Animation

As most readers are no doubt aware, it is possible to project three-dimensional images onto a two-dimensional device (such as a television screen) in such a way as to provide the illusion of a true three dimensional image. Various three-dimensional animation packages are available on the market today, each of which has its own method of accomplishing this. One such package is the subLOGIC assembly language animation package.

The programmer creates a scene by specifying start points, end points, continue points, rays, and points on an x,y,z coordinate field. The x,y,z coordinates can extend

over a +/- 32767 range. In addition, the programmer also specifies the external location of the user's "eye" by providing values for the parameters "pitch", "bank", and "heading". By programmatically altering these parameters, the programmer-defined image can appear to rotate in space.

Scenes are created quite easily via the DEVELOP program provided with the package. Once a scene has been created, it is named and saved as a binary file (therefore taking up little space).

Due to the built-in high speed point generator and screen-erase capabilities, the animation created by programmatically changing pitch, bank, and heading is relatively smooth. However, provided with the package, is the facility for creating much smoother animation sequences via the alternate screen selection technique described above. Unfortunately, 16 kilobytes of RAM is then used up for graphics display, and when one considers that an additional 8 kilobytes is required for the subLOGIC driver program, an additional 10 kilobytes is occupied by DOS, and that the programmer-generated scene and (perhaps) shape tables will occupy still more RAM, there is clearly not much room left for the main program itself!

# APPENDIX III: COURSEWARE EVALUATION INSTRUMENTS

MICROCOMPUTER COURSEWARE

MATERIALS ANALYSIS FORM

TITLE: _____

AUTHOR: _____

PUBLISHER: _____

COPYRIGHT DATE: _____

PREPARED BY: JEM Research in cooperation with the B.C. Ministry of Education

Analyst's Name/Number:

Date:_____19__. Hours Spent Analyzing:__.

SECTION I: Identification and Background

A. PROGRAM TITLE (including diskette and/or series title):

_____

_____

B. AUTHOR(S) (if available): _____

_____

C. PUBLISHER/DEVELOPER (and distributor, if different from publisher/producer):

_____

_____

D. COPYRIGHT DATE(S): _____

_____

E. OPERATING SYSTEM: _____

F. LANGUAGE REQUIREMENTS: _____

_____

G. MEMORY REQUIREMENTS:

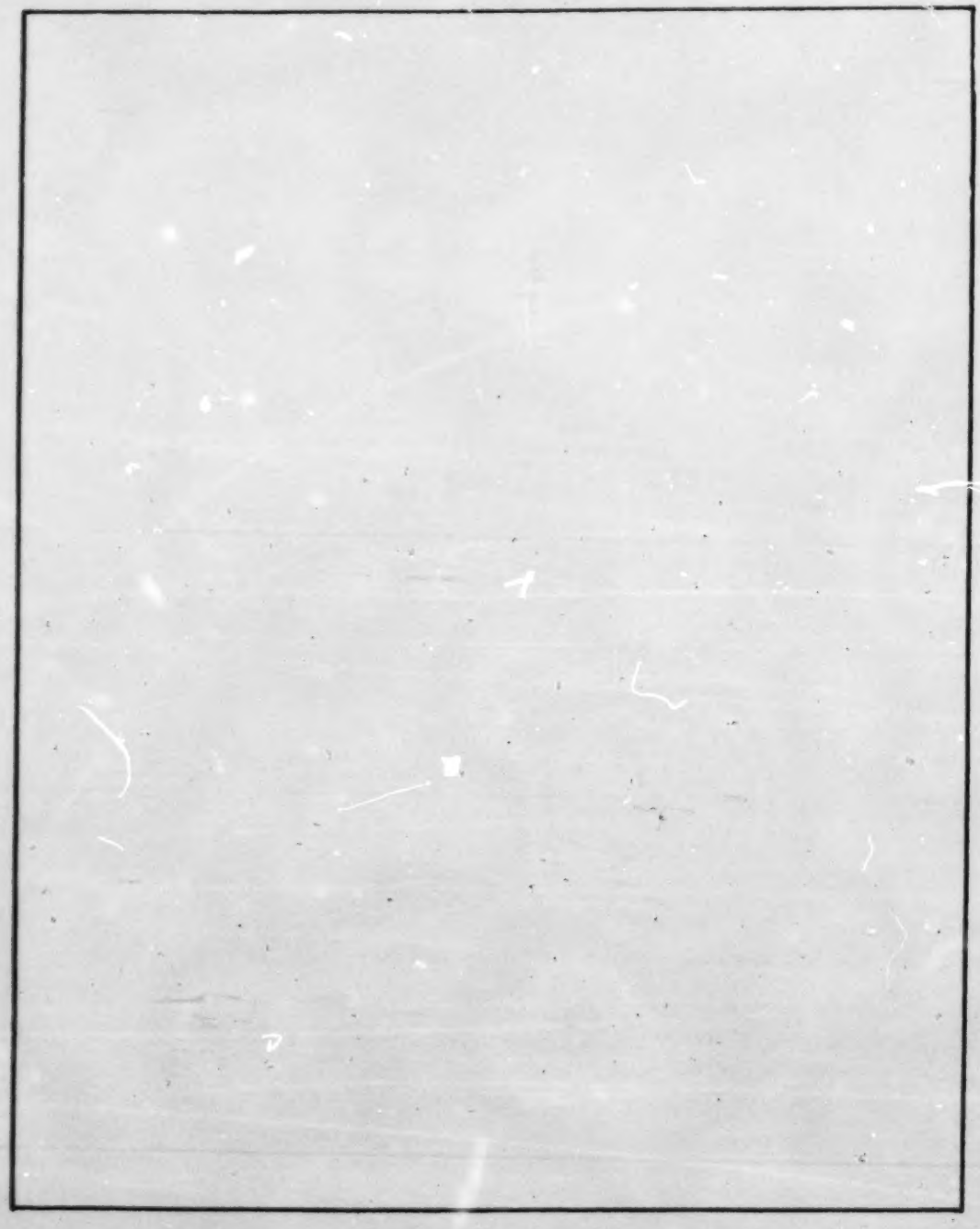
_____

_____

H. PERIPHERAL REQUIREMENTS:

_____

I. COMPONENTS (exact title of each component, with asterisk to indicate items not available for analysis):



J. SUBJECT AREA:

specified in materials:_____; source/pages:_____.

inferred by analyst:_____.

K. CURRICULUM ROLE:

basic:

supplementary:

specified in materials:_____; source/pages:_____.

inferred by analyst:____.

L. INTENDED USERS:

specified in materials:_____; source/pages:_____.

inferred by analyst:____.

M. PRODUCT DEVELOPMENT:

Summarize any comments made by the developer on product development, field testing, product revisions etc.

NOTE: In all sections to follow, be sure to comment on <u>both</u> the program(s) and the documentation.

## SECTION II: Instructional Design

### A. Intents

DEVELOPER'S RATIONALE

(Describe the developer's reasons for developing the program. If you quote, quote accurately.) specified by developer:_____; source/pages:_____; inferred by analyst:____

You may wish to report on assumptions made about:

- -specific content
- -subject matter approach
- -societal need
- -system of management
- -approach
- -skill building
- -"affect", e.g., interest learners, improve attitudes, change moral climate
- -special learners
- -suitability for microcomputer application

LEARNER GOALS

(Give examples of goal statements, or statements from which goals can be inferred. If you quote, quote accurately.) specified by developer:_____; source/pages:_____; inferred by analyst:____; left to user:____.

You should report on:

-explicitness
-learning domain emphasis, e.g., cognitive, affective, psychomotor.
-achievement expectations, e.g., fixed, ordered by priority, variable.

LEARNER OBJECTIVES

(Give examples of objectives, or statements from which objectives can be inferred. If you quote, quote accurately.) specified by developer:_____; source/pages:_____; inferred by analyst:_____; left to user____.

You should report on:

-explicitness
-learning domain emphasis, e.g., cognitive, affective, psychomotor
-specificity
-achievement expectations, e.g., fixed, ordered by priority, variable, number of objectives is specified

B. Contents

ENTRY COMPETENCIES

(Discuss and give examples of ENTRY COMPETENCIES.) specified by developer:_____; source/pages:_____; inferred by analyst:_____.

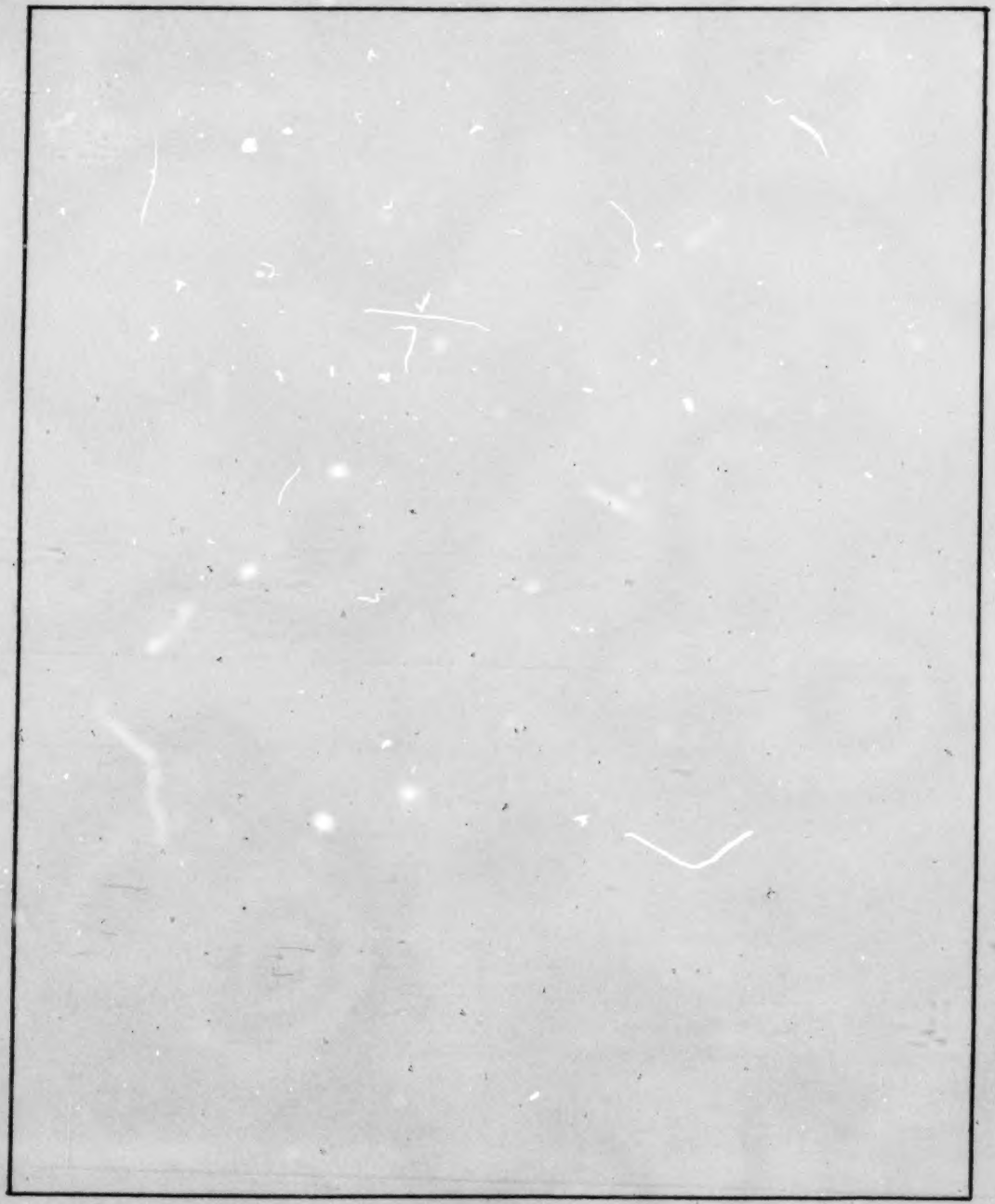
You should consider:

-if nc special competencies are necessary
-if there are competencies learner needs to bring to materials that developer does not identify
-if developer identifies competencies and provides readiness materials
-if developer identifies competencies and provides materials for various entry levels
-if familiarity with a microcomputer is required

SCOPE: Topics Covered and Characteristics

SCOPE: Topics Covered

(Give a comprehensive listing of topics and skills covered by each level.)



SCOPE: Characteristics

(Discuss characteristics of the SCOPE of the program(s).)

Where appropriate you should report on:

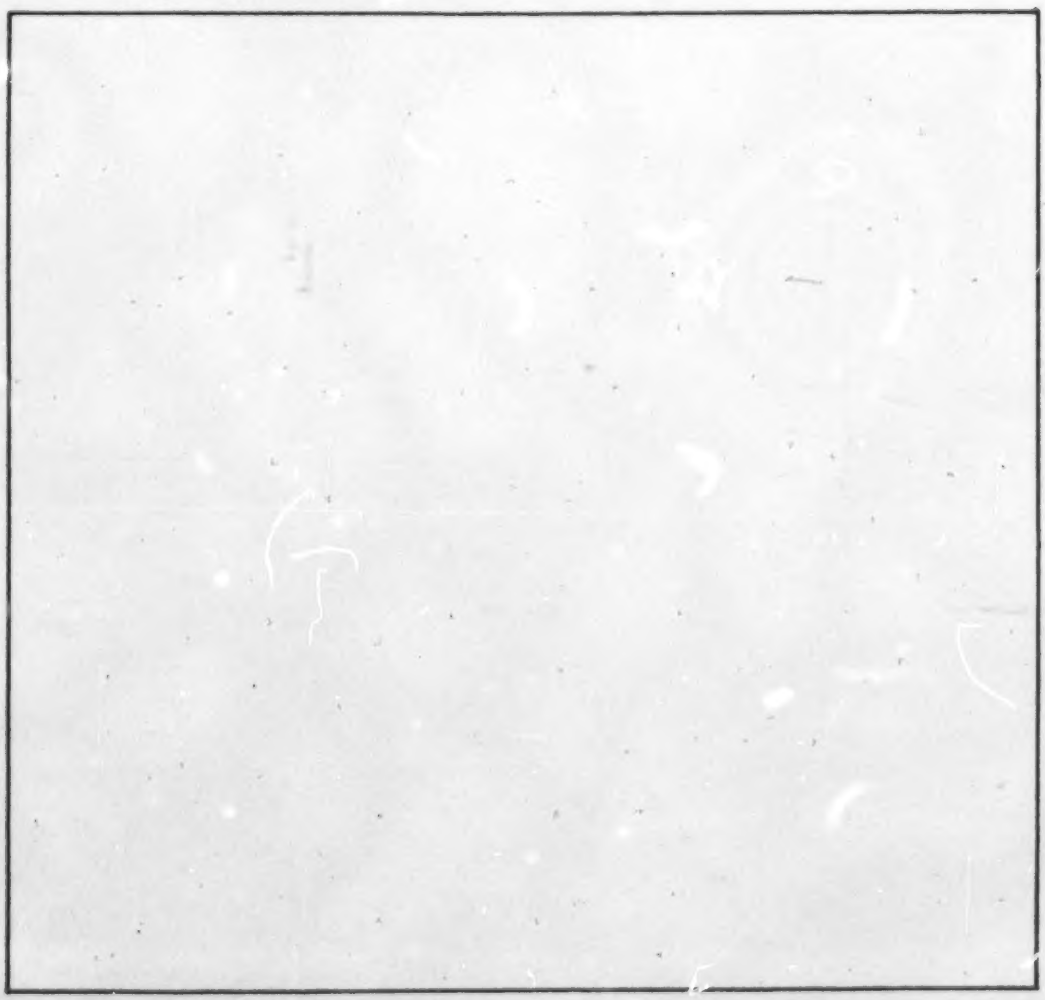
- -rationale for content selection
- -range of total package
- -range within sections or units
- -extent of subject area treatment for total package
- -extent of topic (unit) treatment
- -coverage expectations
- -supporting materials
- -dependence on supporting materials

SEQUENCE

(Discuss the SEQUENCE of the program.)

You should report on:

-rationale for content arrangement
-entry into SEQUENCE for total package
-entry within sections or units
-routing through total package
-routing within sections or units
-selector of SEQUENCE
-branching
-recycling
-ease of routing



C. Methodology

PRESENTATION

(Describe explicitness and sources of the developer's description.)
specified by developer:_____; source/pages:_____; left to user:_____.

CHARACTERISTICS: Description

(Examine the materials and thoroughly describe a typical lesson. If there are different types of lessons or if lessons vary from level to level, you can describe more than one lesson or compare lessons, or both.)

You should comment on:

- -learning approach i.e., drill and practice, tutorial, simulation, dialogue, problem solving, game
- -learning setting i.e., latitude, role of the teacher, role of the learners, grouping arrangements, time, pacing, decision maker
- -learning activities (list them and include explicitness, modes of presentation or interaction, participants in the activities, decision maker)

TEACHER TRAINING/PREPARATION

(Describe provisions for TEACHER TRAINING.) specified by developer:_____; source/pages:_____; inferred by analyst:_____.

You should report on:

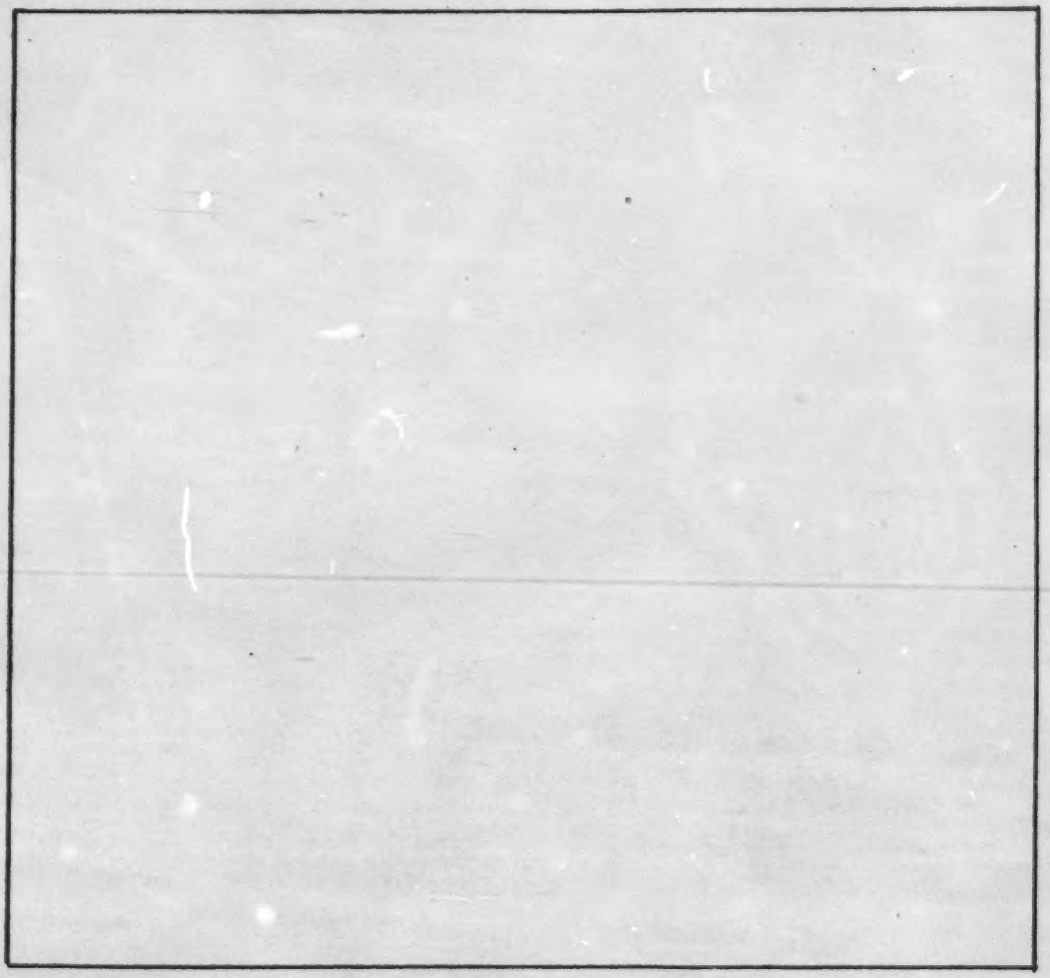
-explicitness
-activities
-availability/format e.g., workshops at publisher's/distributer's or user's expense, training materials at publisher's/distributer's or user's expense
-occasions when teacher preparation is needed

EVALUATION

(Describe the EVALUATION MEANS OR APPROACHES.)

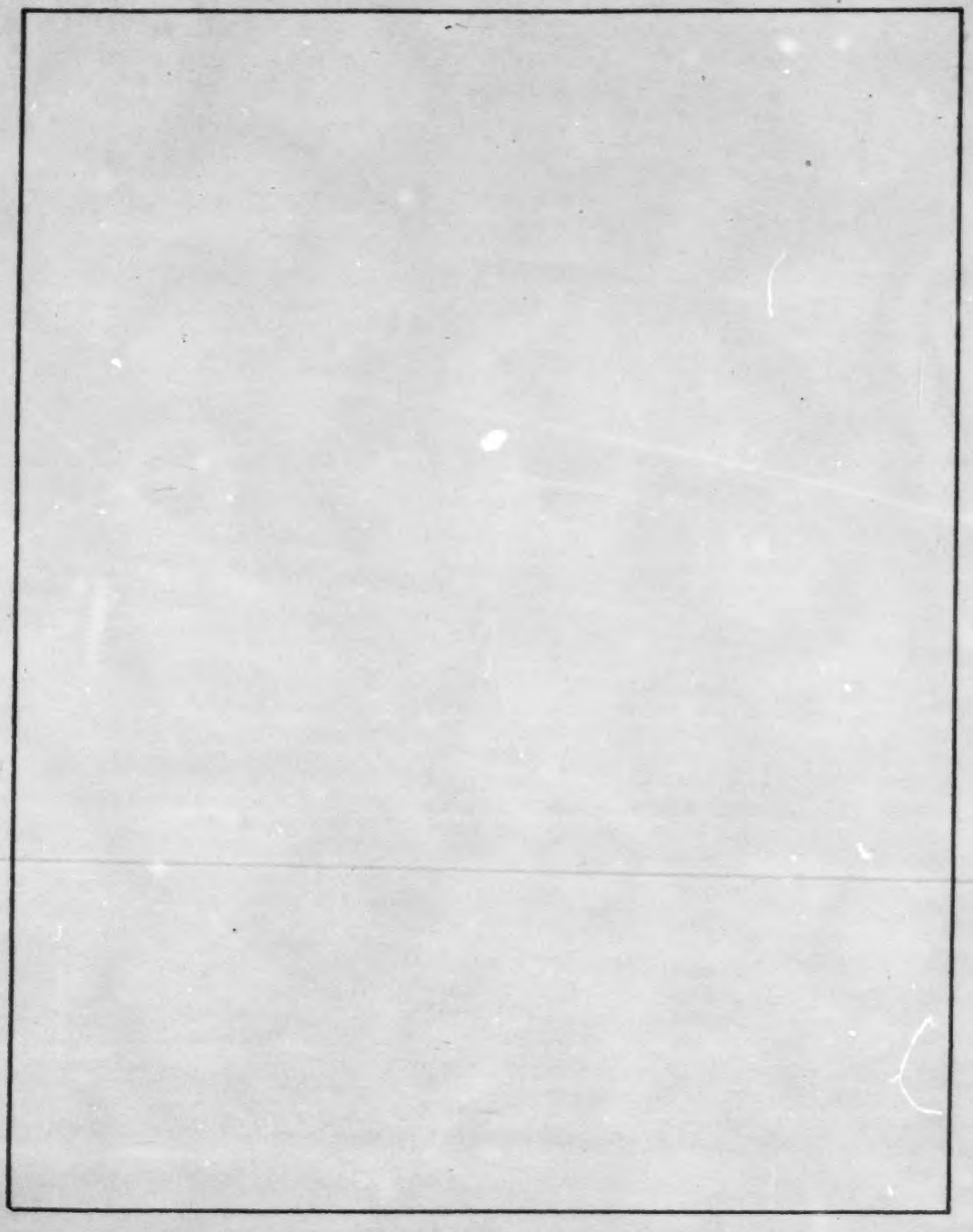
You should report on:

-Presentation (i.e. are the means of evaluation left to the teacher?)
-Purpose (i.e. determination of preinstructional status, pretest for placement in sequence, diagnosis of specific difficulties, provision of formative or corrective feedback to guide instruction, assessment of learning outcome, summative evaluation of achievement)
-Content Focus (i.e. learning domain emphasis)
-Basis of Evaluation (i.e. norm or criterion referenced)
-Format (i.e. fill in blanks, multiple choice etc.)



**Congruence**

**(Discuss the "goodness of fit" between Intents/Contents and Methodology/Evaluation.)**



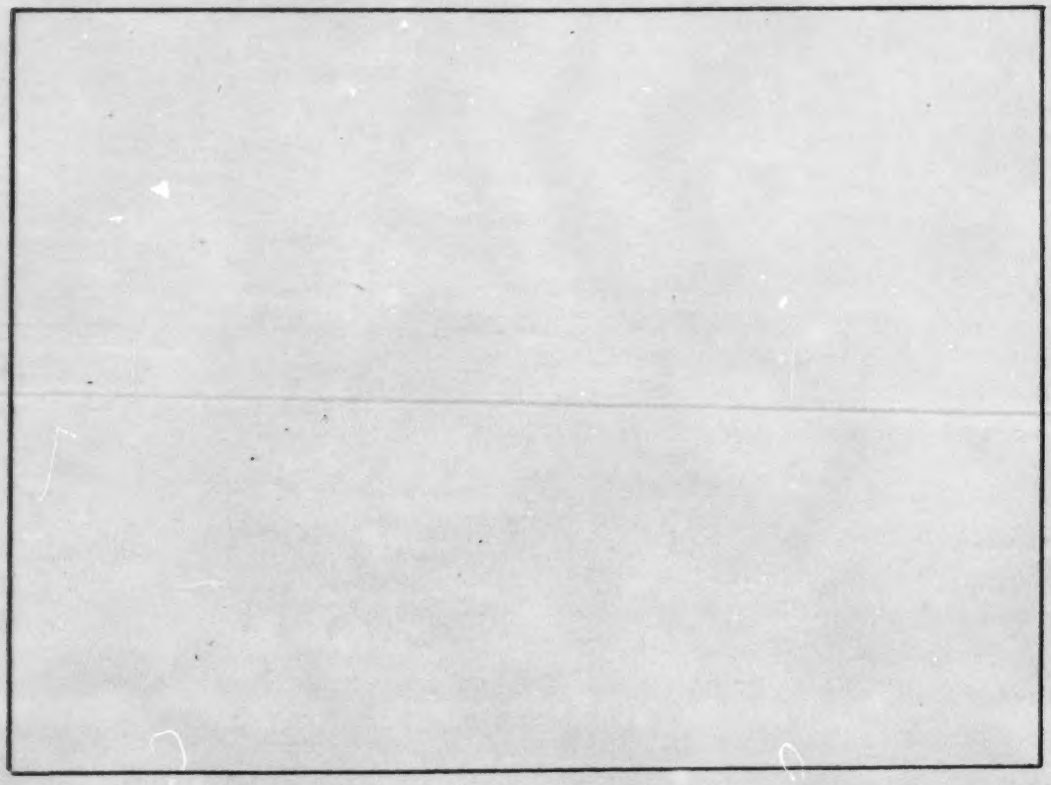
## SECTION III: Technical Considerations

**Output:**

(Describe the ways in which materials are presented.)

You should report on:

- -Ease of loading of cassette tapes (if applicable)
- -Method of output (i.e. paged or scrolled)
- -Amount of user control over pacing
- -Effectiveness of use of space over entire screen/page
- -Suitability of perceptual impact
- -Language usage (i.e. correctness of grammar, spelling, hyphenation and punctuation)
- -Use of colour, graphics and sound
- -Use of devices to get and maintain interest (i.e. humour, surprise, personalized dialogue)

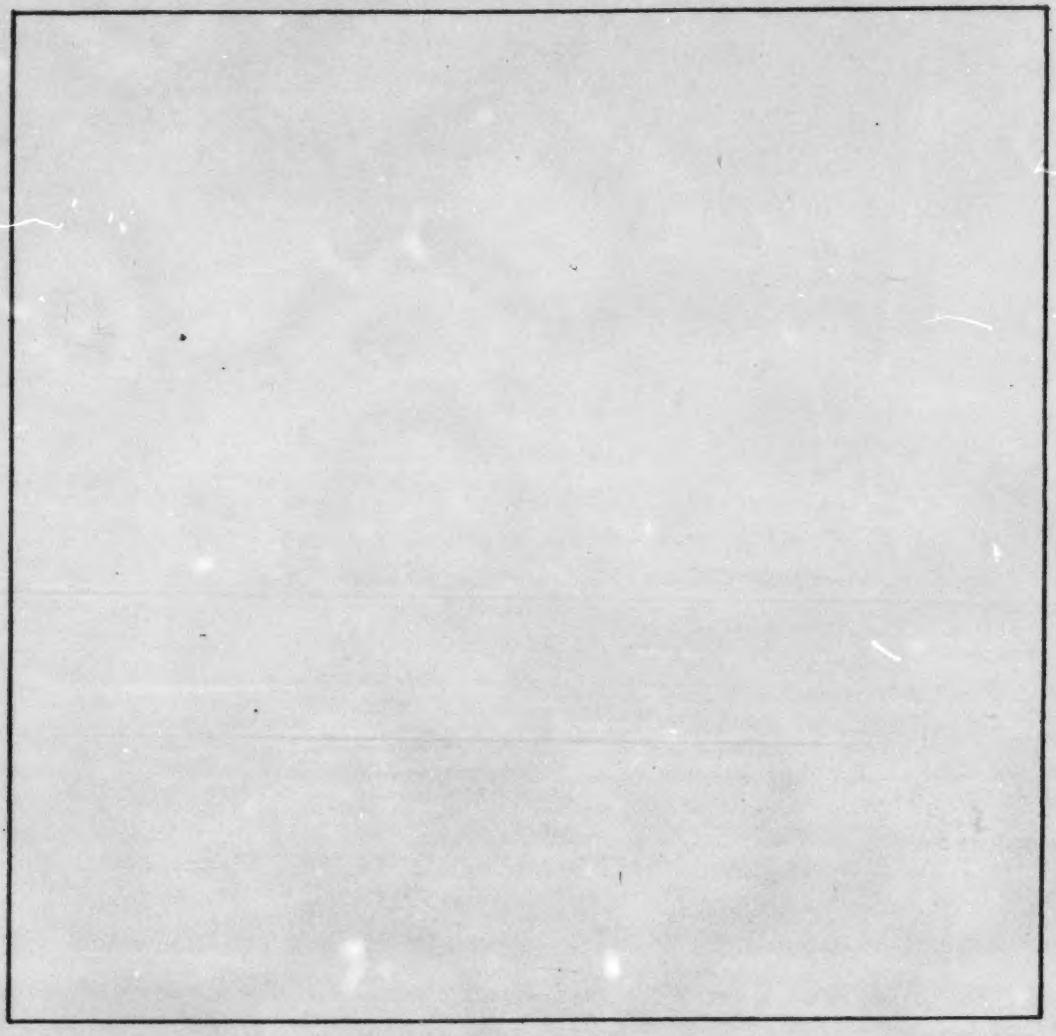


Input:

(Describe the ways in which users input information.)

You should report on:

-Ability of program to deal with inappropriate entries (synonyms etc.)
-Adequacy of instructions
-Ability of user to respond with common symbols and ways of using them (e.g. left to right entry of sums)
-Ability of program to accept abbreviations for common responses
-Amount of input required (i.e. is user generally active or passive?)



## SECTION IV: Readability

**Surface Measures**

**Formula:**

(Calculate a readability score. Indicate sample scores and formula used.)

Sample #1
Ref.

Sample #3
Ref.

Sample #2
Ref.

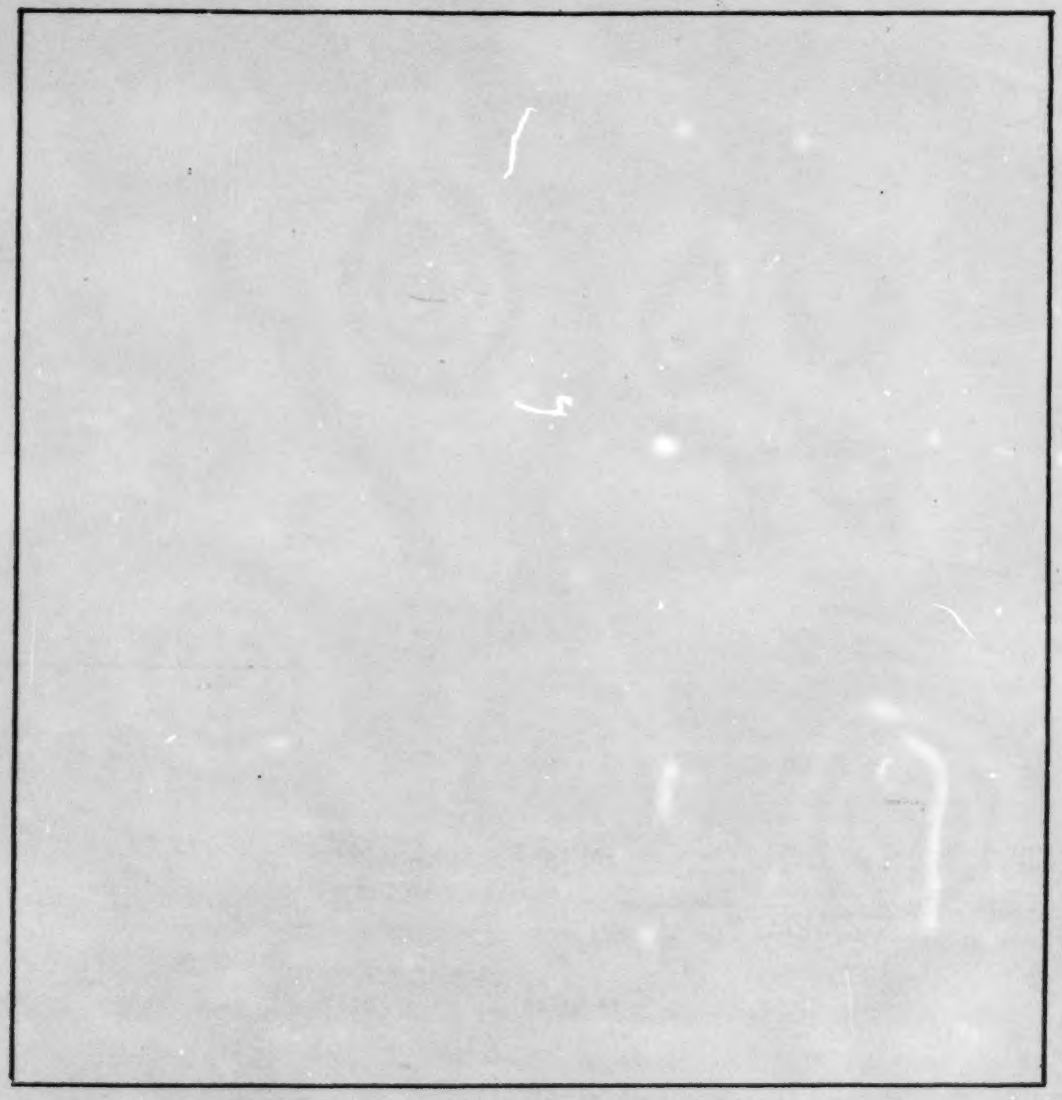
Sample #4
Ref.

Other Considerations:

(Describe other factors which might effect the readability.)

You should comment on:

-Print size
-Print clarity
-Word Emphasis i.e. Flashing display, Underlining, Inverse display
-Graphic support to illustrate text
-Presence of definitions within text



**Deep Measures**

**Idea Load:**

**(Describe how the idea load effects readability.)**

**You should comment on:**

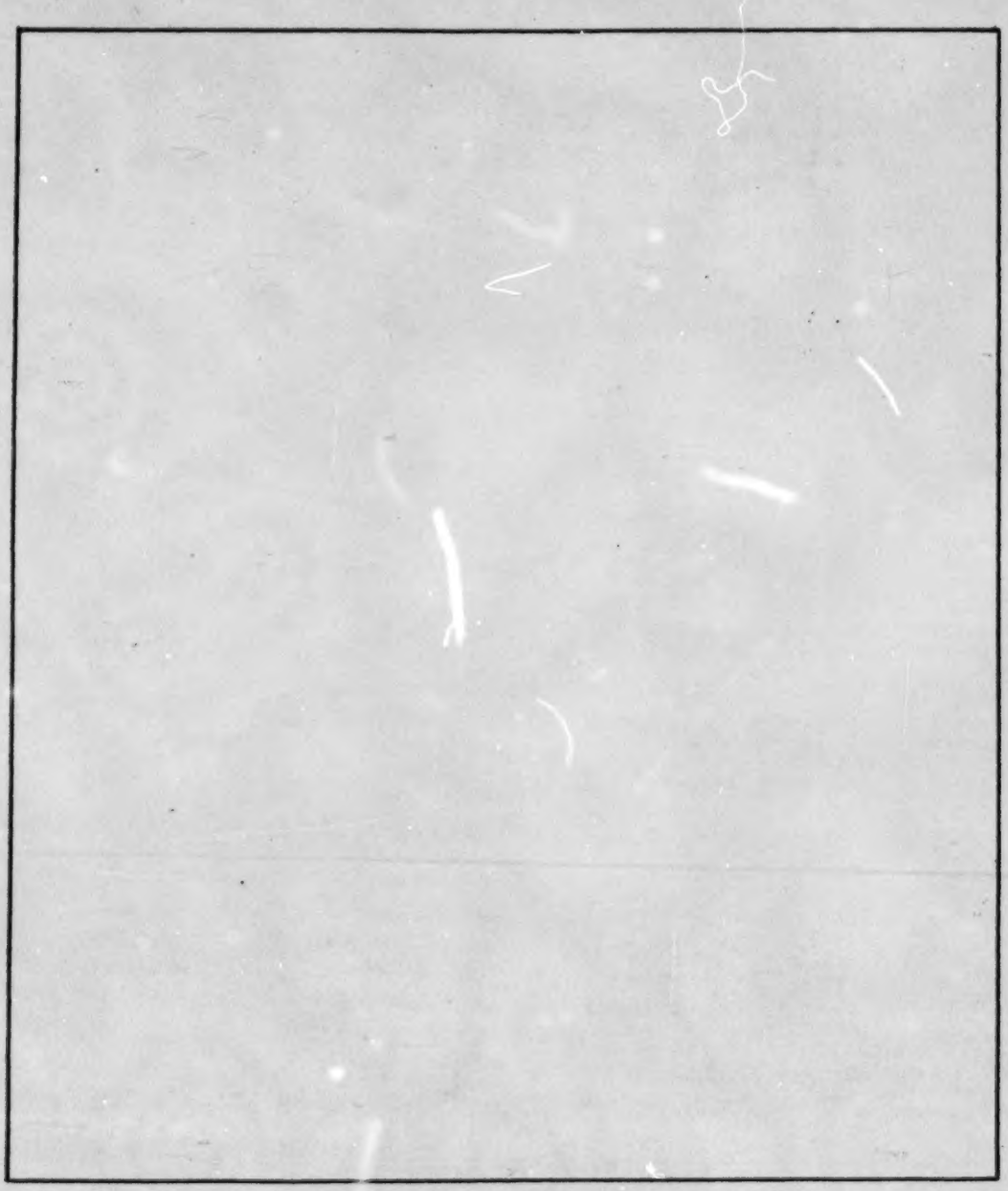
- **-Concrete/Abstract ideas**
- **-Repetition of ideas**
- **-Sequencing**
- **-Number of ideas:**
  - **(i) Overall**
  - **(ii) Per page or screen**

Vocabulary:

(Describe how the vocabulary effects readability.)

You should comment on:

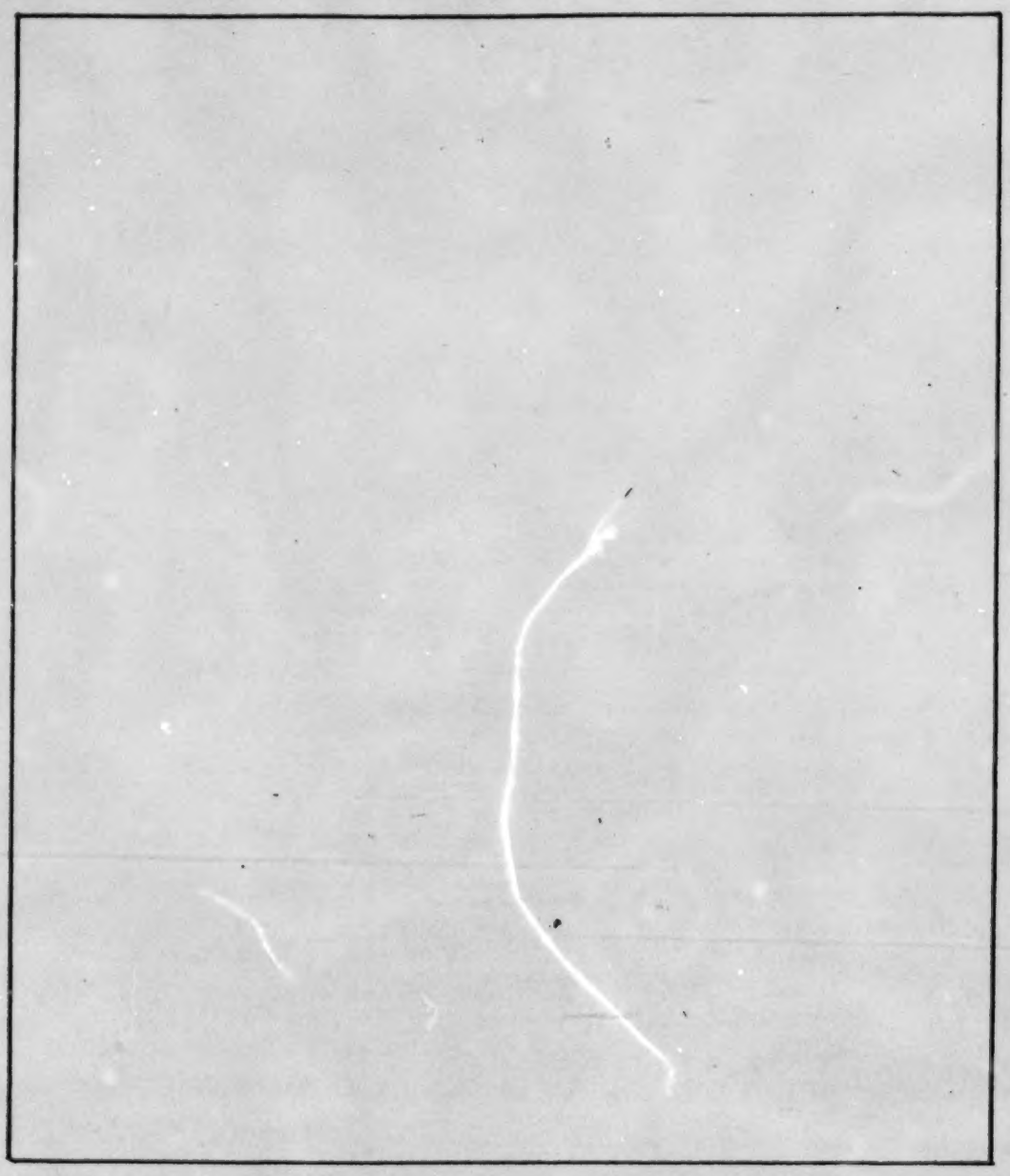
-Technical vocabulary
-Definitions (presence of textual or graphic clarification)



## SECTION V: Other Considerations

**PHYSICAL DESCRIPTION: Packaging**

(Briefly describe the program components giving as clear an idea as possible of size, binding (when appropriate, e.g., consumables), number of pages of print materials, number of diskettes or cassettes, approximate running time.)



ACCURACY OF CONTENT

(Describe the type and extent of inaccuracies or distortions if any are noted. Include source and pages. If you quote, quote accurately.)

Among other possibilities you can classify inaccuracies as:

-factual errors
-omissions
-inappropriate placement of topics
-unsuitable reading level
-misleading or irrelevant diagram charts or tables
-computational errors
-inaccurate responses in answer keys
-incomplete index, glossary or table of contents
-general editorial slovenliness
-content unsuitable or otherwise inappropriate

Extent of inaccuracies can be described as:

-not discernible
-serious enough to interfere with instruction
-errors need to be corrected
-errors render contents useless
-some errors not serious enough to interfere with effective instruction

CURRENCY OF CONTENT

(Describe and give examples of why the content of the program is current or not. Include source and pages. If you quote, quote accurately.)

Among other possibilities, currency considerations may be described as:

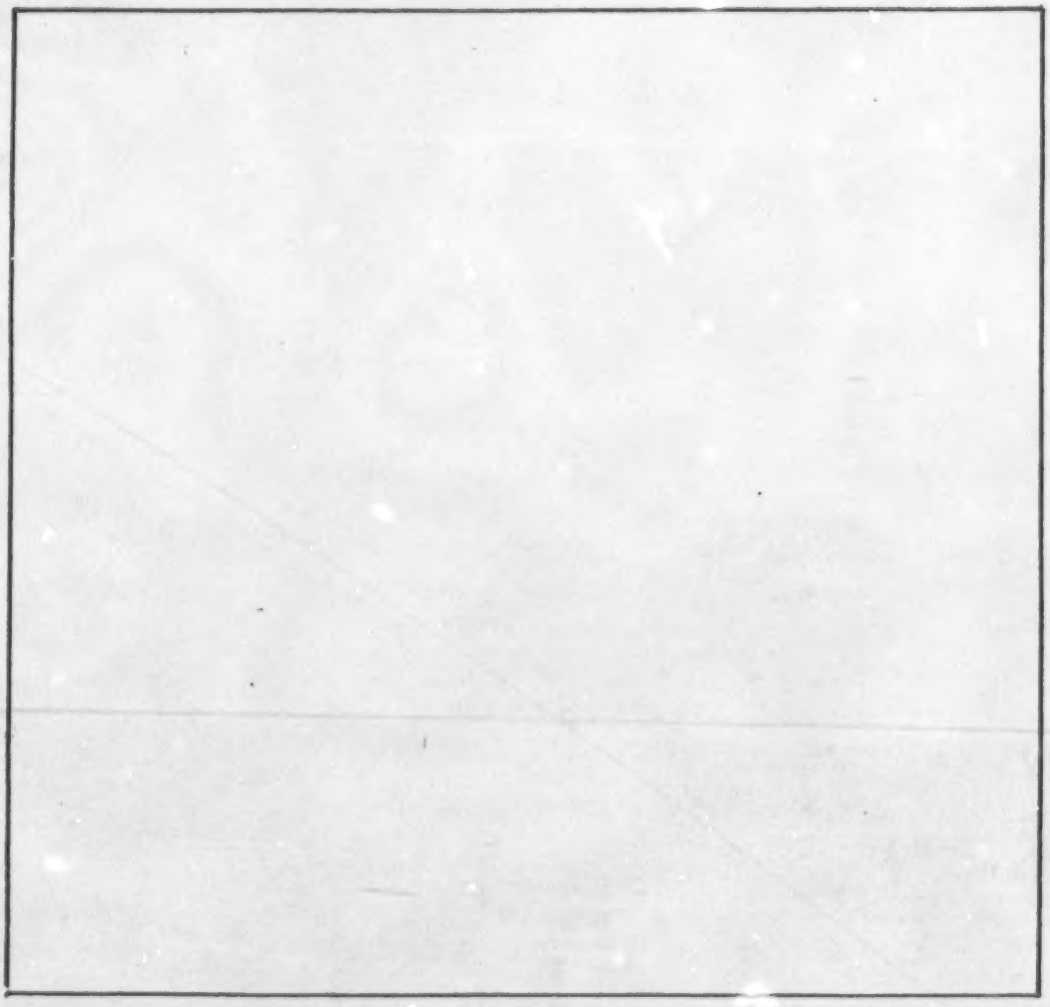
-dated in treatment of conventional topics
-lacking currency in language usage
-lacking currency in physical design
-including most currently treated topics
-reflecting in topics major changes in the discipline
-following in topics of current fads
-not being acceptable topics for study in present social climate

Social Considerations

(Analysts will recognize that these comments must necessarily be subjective, based on their experience but they should identify anything which they feel might be questioned.)

You should comment on the following where appropriate:

-Language usage (i.e. profanity, graphic descriptions, religious allusions)
-Ethnic references
-Sexual references
-Violence
-Racial references
-Religious references
-Political references



MEDIA APPROPRIATENESS

Comment on the appropriateness of the media, e.g., visuals, and the extent to which these elements support the content (and learning activities) of the text or narration to which they are related.

CURRICULUM MATCH

(Describe the relevance of the materials to the local curriculum) - where possible, indicate specific curriculum matches.

**ANALYST'S COMMENTS**

(Present your general critique of the program, touching on its unique features and similarity to other programs in the field, possible limitations in terms of intended users and settings, external constraints, and possible problems in implementation and acceptance. Give bases for judgements where feasible, and illustrate.)

## SCHOOL MICROWARE EVALUATION FORM

(This form may be copied as needed; for additional information, see School MicroWare Vol. 1 No. 3

Your Name_____Organization_____Position_____

Product Name_____Supplier_____No. of Progs. Under This Name_____Price $_____

Applicable School Dept/Subj/Topic _____

INSTRUCTIONS - For open ended items, supply all information requested in blanks provided if possible; use extra sheets if necessary. For objective items (those with blanks to left), enter a number in the blank to indicate the extent to which the program fulfills the description in the item, as follows: 2 - Completely, 1 - Partially, 0 - Not at All. If the item is not applicable to the program, enter N/A. If the item is unclear, enter U. Elaborate on answers as necessary in Comments section at end or on extra sheets, giving item numbers.

OVERVIEW - Describe the program briefly in terms of its goals and what it does to achieve them (no evaluation here).

_____

_____

PRELIMINARY CONSIDERATION - Assuming that this program contributes to the teaching of one or more topics, is that topic one which is or should be taught in today's schools? Yes No If not, give your reasons for this answer in the Comments section at the end of the form and omit the balance of the questionnaire.

DOCUMENTATION - List materials accompanying the program, e.g., teachers guide, student workbook.

_____ 1. Indicate types of information included.
_____ a. Suggested course/subject, grade levels.
_____ b. Goals.
_____ c. Performance objectives.
_____ d. Suggested teaching strateg(ies).
_____ e. Correlation with standard texts.
_____ f. Prerequisites for use of program.
_____ g. Student exercises, teacher answers.
_____ h. Operating instructions.
_____ i. Listing and sample runs of program(s).
_____ j. If a simulation, description of the model used.
_____ k. Suggested topics for follow-up discussions.
_____ l. Suggested references/activities for follow-up.
_____ 2. The documentation is written clearly.
_____ 3. If a workbook is included, the format and content are appropriate.

INSTRUCTIONS GIVEN TO USER BY PROGRAM

1. The instructions are adequate regarding:
_____ a. The instructional task to be performed.
_____ b. Details of how to interact with the program.
_____ 2. User has the option of skipping instructions if already known.

STUDENT-COMPUTER DIALOG

_____ 1. Output is displayed screen by screen (paged) rather than scrolled.
2. If output is paged:
_____ a. User has control over continuing to the next page.
_____ b. Amount of information in each page is appropriate.
_____ c. The perceptual impact (amount of type and lines) is suitable.
_____ 3. Output is spaced and formatte1 so as to be easily readable.
_____ 4. Language is well suited to most students' reading ability.
_____ 5. Uses correct grammar,spelling, hyphenation and punctuation.
_____ 6. Any grid or coordinate system used is consistent with common conventions.
_____ 7. Students can respond with common symbols & ways of using them, e.g., right to left entry of sums.
_____ 8. Accepts abbreviations for common responses.
_____ 9. Provides for individual needs, e.g., opportunity to work with harder or easier material.
_____ 10. Dialog is personalized, i.e., makes appropriate use of student names.
_____ 11. Uses devices to get & maintain interest, e.g., variation of computer responses, humor, pace change, surprise.
_____ 12. Makes good use of any special features of computer:
_____ a. Graphics
_____ b. Color
_____ c. Sound
_____ 13. Reinforcing responses (indications of right, wrong, etc.) are appropriate.
_____ 14. The number of wrong answers allowed is reasonable.
_____ 15. Responds appropriately if allowed number of wrong answers is exceeded.
_____ 16. Provides opportunity to get help if difficulty is encountered.
_____ 17. Minimizes bad entries via devices such as objective formats (multiple choice,etc).
_____ 18. Deals well with inappropriate entries, i.e., response to typing errors, etc., is intelligible and useful.
_____ 19. Required entries are within students' capabilities (esp. typing, vocabulary).
_____ 20. Reports student performance periodically and at end of session.

MISCELLANEOUS CONCERNS

_____ 1. If a simulation, the program gives a sufficiently accurate representation of the situation simulated.
_____ 2. The concepts and vocabulary required to use the program are reasonable.
_____ 3. Operates properly and is free of bugs.
_____ 4. Is well structured and documented internally to facilitate any necessary debugging/modification.

COMMENTS - Please use this space and additional sheets as necessary to provide any other information which you believe would help someone who was considering acquiring the program being reviewed. In particular, indicate what you like least and most about the program. Also, list any changes which should be made.)

_____

_____

# REFERENCES

## References

Aiken, R. M., & Braun, L. Into the 80's with Microcomputer - Based Learning. Computer, 1980, 13(7), 11-16.

Alessi, S. M., & Dennis, J. R. Designing Instruction for Teaching with a Computer. Illinois Series on Educational Application of Computers, 1979, No. 3e.

Atherton, R. Microcomputers, Secondary Education and Teacher Training. British Journal of Educational Technology, 1980, 10(3), 198-216.

Becker, S. A. Legal Protection for Computer Hardware and Software. Byte, 1981, 6(5), 140-146.

Bishop, R. D. Mix It Up On Your Apple. Kilobaud Microcomputing, 1981, 5(7), 108-114.

Blaschke, C. L. Microcomputer Software Development for Schools: What, Who, How? Educational Technology, 1979, 19(10), 26-28.

Bloom, B. S., et al. Taxonomy of Educational Objectives: Cognitive Domain. New York: Longman, 1956.

Bork, A. Educational Technology Centre at the University of California. Computer Assisted Learning, Lewis, R., & Tagg, E. D. (Ed.), New York: North - Holland Publishing, 1980, 37-53.

Bowles, K. L. Microcomputer - Based Mass Education. Journal of Personalized Instruction, 1978, 3(3), 151-156.

Caldwell, R. M. Guidelines for Developing Basic Skills Instructional Materials for Use with Microcomputer Technology. Educational Technology, 1980, 20(10), 7-12.

Chambers, J. A., & Sprecher, J. W. Computer Assisted Instruction: Current Trends and Critical Issues. Communications of the Association for Computing Machinery, 1980, 23(6), 332-342.

Clement, F. J. Affective Considerations in Computer - Based Education. Educational Technology, 1981, 21(4), 28-32.

Cole, B. C. Computer Language Roundup. Interface Age, 1981, 6(6), 75-83.

Cole, H. P. Piaget's Theory of Cognitive Representation and Brain Hemisphere Functioning. Educational Technology, 1980, 20(12), 38-40.

Dence, M. Toward Defining a Role of CAI: A Review. Educational Technology, 1980, 20(11), 50-54.

Dennis, J. R., Muiznieks, V. J., & Stewart, J. T. Instructional Games and the Computer - Using Teacher. Illinois Series on Educational Application of Computers, 1979, No. 9e.

Denis, J. R. The Question - Building Block of Teaching with a Computer. Illinois Series on Educational Application of Computers, 1979, No. 4e.

Denis, J. R. Tutorial Instruction on Computer. Illinois Series on Educational Application of Computers, 1979, No. 6e.

Dijkstra, E. GOTO Statement Considered Harmful. Communications of the ACM, 1968, 11(3), 147-148.

Donoian, G. A Paradigm for Computer - Based Instruction: Renaissance of the Affective Domain. Association for Educational Data Systems, 1981, Convention Proceedings, 84-88.

Eisele, J. E. A Case for Computers in Instruction. Journal of Research and Development in Education, 1980, 14(1), 1-8.

Gaede, O. F. Computers in Science Education. Illinois Series on Educational Application of Computers, 1979, No. 17e.

Gagne, R. M., & Briggs, L. J. Principles of Instructional Design. New York: Holt, Rinehart and Winston, 1974.

Garraway, H. It's as Easy as...1,2,3. Instructional Innovator, 1980, 25(6), 20-21.

Gerhold, G. Teacher Produced CAI. Computer Assisted Learning, Lewis, R. & Tagg, E. D. (Ed.), New York: North - Holland Publishing, 1980, 15-25.

Gershman, J., & Sakamoto, E. Computer - Assisted Remediation and Evaluation: A CAI Project for Ontario Secondary Schools. Educational Technology, 1981, 21(3), 40-43.

Gleason, G. T. Microcomputers in Education: The State of the Art. Educational Technology, 1981, 21(3), 7-18.

Grogono, P. Programming in Pascal. Reading, Massachusetts: Addison-Wesley, 1978.

Gropper, G. L. Systematic Course Design. Association for the Development of Computer-Based Instructional Systems News, 1980, 12(6), 42-44.

Hallworth, H. J., & Brebner, A. Computer Assisted Instruction in Schools: Achievments, Present Developments and Projections for the Future. Calgary: Faculty of Education Computer Applications Unit, 1980.

Hausmann, L. J. Effective Feedback in Computer Assisted Instruction. Association for Educational Data Systems, 1981, Convention Proceedings, 38-41.

Hartman, J., & Garnett, D. A Structured Approach to the Development of Computer Assisted Instruction. Association for Educational Data Systems, 1981, Convention Proceedings, 119-124.

Hirschbuhl, J. J. Hardware Considerations for Computer Based Education in the 1980's. Journal of Research and Development in Education, 1980, 14(1), 41-56.

Hofstadter, D. R. Godel, Escher, Bach: An Eternal Golden Braid, New York: Vintage Books, 1980.

Huntingdon, J. F. Computer Assisted Instruction Using BASIC, New Jersey: Educational Technology Publications, 1979.

Jelden, D. L. A CAI "Coursewriter" System for the Microcomputer. Association for Educational Data Systems Journal, 1981, 14(3), 159-168.

Kay, A. C. Microelectronics and the Personal Computer. Scientific American, 1977, 273(3), 231-244.

Kearsley, G. P. Some 'Facts' About CAI: Trends 1970 - 1976. Journal of Educational Data Processing, 1976, 13(3), 1-11.

Kehrberg, K. T. Microcomputer Software Development: New Strategies for a New Technology. Association for Educational Data Systems Journal, 1979, 13(1), 103-110.

Koch, W. J. Basic Facts About Using the Computer in Instruction. Education Digest, 1973, 38(7), 28-31.

Krueger, W. C. F. The Effect of Overlearning on Retention. Journal of Experimental Psychology, 1929, 12, 71-78.

Kulhavy, W. K. Feedback and Response Confidence. Journal of Educational Psychology, 1976, 68(5), 522-528.

Livingston, A. Instructional Development: Establishing the Running Game. Educational Technology, 1980, 20(11), 20-25.

Lower, S. K. Compilers and Interpreters. Micro-scope, 1981, 2(4), 20-21.

Magel, K. Software Engineering Principles for Courseware Development. Association for Educational Data Systems Journal, 1981, 13(2), 144-155.

Mager, R. F. Preparing Instructional Objectives. Belmont, CA: Pearon Publishers, 1962.

Moore, M. V., & Nawrocki, L. H. The Educational Effectiveness of Graphic Displays for Computer Assisted Instruction. Alexandria, Virginia: U.S. Army Research Institute for the Behavioral Sciences, 1976.

Nelson, T. Interactive Systems and the Design of Virtuality. Creative Computing, 1980, 6(11), 56-62.

Nelson, T. (Ed.). Symposium on Actor Languages. Creative Computing, 1980, 6(10), 61-86.

Peters, H. J., & Johnson, J. W. Author's Guide: Design, Develoment, Style, Packaging, Review. University of Iowa: CONDUIT, 1978.

Pogue, R. E. The Authoring System: Interface Between Author and Computer. Journal of Research and Development in Education, 1980, 14(1), 57-67.

Postman, L. Retention as a Function of Degree of Overlearning. Science, 1962, 135, 666-667.

Rankin, R. J., & Trepper, T. Retention and Delay of Feedback in a Computer - Assisted Instructional Task. Journal of Experimental Education, 1978, 46(4), 67-70.

Robinson, M. On Effective Documentation. Creative Computing, 1980, 6(11), 30-31.

Roblyer, M. D. Instructional Design vs. Authoring of Courseware: Some Crucial Differences. Association for Educational Data Systems, 1981, Convention Proceedings, 243-247.

Schuyler, J. A. Programming Languages For Microcomputer Courseware. Educational Technology, 1979, 19(10), 29-35.

Shay, C. Simulations in the Classroom: An Appraisal. Educational Technology, 1980, 20(11), 26-31.

Silverman, R. E. Psychology, New York: Appleton - Century - Crofts, 1971.

Spitzer, D. R. Motivational Design: Toward a New Educational Technology? British Journal of Educational Technology, 8(1), 63-69.

Staats, C. K., & Staats, A. W. Attitudes Established by Classical Conditioning. Journal of Abnormal and Social Psychology, 1957, 54, 74-80.

Stewart, J. T. Drill and Practice on a Computer. Illinois Series on Educational Application of Computers, 1979, No. 7e.

Sturges, P. T. Verbal Retention as a Function of the Informativeness and Delay of Informative Feedback. Journal of Educational Feedback, 1969, 60(1), 11-14.

Sugarman, R. A Second Chance for Computer - Aided Instruction. IEEE Spectrum, August, 29-37.

Tait, K., Hartley, J. R., & Anderson, R. C. Feedback Procedures in Computer - Assisted Instruction. Journal of Educational Psychology, 1974, 43(2), 161-171.

Zaks, R. An Introduction to Pascal. Berkeley: Sybex Press, 1981.